Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Založnik

Razvoj spletne aplikacije za upravljanje odnosov s strankami na področju nepremičnin

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Obstaja veliko različnih sistemov za upravljanje odnosov s strankami, ki so primerni za različna področja. Ti sistemi pa so zelo splošni in pogosto težko prilagodljivi za specifična področja, kot je na primer področje posredovanja nepremičnin. V diplomski nalogi tako razvijte namenski sistem za upravljanje odnosov s strankami, ki bo prilagojen področju posredovanja nepremičnin. V ta namen razvijte primeren podatkovni model in razvijte spletno aplikacijo za upravljanje odnosov s strankami. Pri razvoju aplikacije izberite ustrezne tehnologije na strani strežnika in na strani odjemalca. Glede na samo naravo dela na področju posredovanja nepremičnin, ki zahteva tudi mobilnost uporabnikov sistema, pa poskrbite tudi za primerno delovanje in prikaz odjemalskega dela na mobilnih napravah.

Ob tej priložnosti se iskreno zahvaljujem mentorju doc. dr. Alešu Smrdelu za čas in pomoč pri pisanju diplomskega dela. Posebna zahvala gre tudi staršem in vsem ostalim, ki so mi v času študija stali ob strani ter me podpirali.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CRM	customer relationship management	upravljanje odnosov s strankami
SPA	single-page application	eno-stranska aplikacija
MVC	model-view-controller	model-pogled-nadzornik
REST	representational state transfer	predstavitveni prenos stanja
DRF	django REST framework	django REST ogrodje
DRY	don't repeat yourself	ne ponavljaj se
ORM	object-relational mapping	objektno-relacijsko mapiranje
VCS	version control system	sistem za nadzorovanje verzij
ORDBMS	object-relational database management system	sistem za upravljanje objektno-relacijskih podatkovnih baz
HTML	hyper text markup language	hiper tekstovni označevalni jezik
CSS	cascading style sheets	kaskadne slogovne pole
API	application programming interface	aplikacijski programski vmesnik
DOM	document object model	dokumentni objektni model
IT	information technology	informacijska tehnologija

Povzetek

Naslov: Razvoj spletne aplikacije za upravljanje odnosov s strankami na področju nepremičnin

Avtor: Rok Založnik

Dobri odnosi s strankami so v današnjih ekonomskih razmerah za podjetja zelo zaželjeni, če ne celo nujno potrebni. Podjetja, ki se tega zavedajo vpeljujejo strategijo poslovanja, ki se osredotoča na dobre odnose s strankami, pri tem pa si pomagajo s aplikacijami za upravljanje odnosov s strankami (CRM). Večina obstoječih rešitev je preveč splošnih za praktično uporabo in dobro učinkovitost na bolj specifičnih področjih, kot so npr. nepremičnine. To nas je vodilo v razvoj spletne aplikacije, ki bo področje poslovanja z nepremičninami boljše podprla. Za razvoj smo uporabili Django REST Framework v kombinaciji s PostgreSQL podatkovno bazo na strani strežnika in ogrodjem AngularJS za odjemalski del aplikacije. Prvi del diplomske naloge je namenjen predstavitvi poslovne strategije CRM in predstavitvi tehnologij ter orodij, ki so bili uporabljeni pri razvoju spletne aplikacije. Drugi del naloge opisuje praktičen razvoj spletne aplikacije. V tem delu je opisana arhitektura aplikacije, podatkovni model ter implementacija in funkcionalnosti posameznih delov aplikacije. V zadnjem delu pa so opisane dodane sklepne ugotovitve in ideje za nadaljnje delo.

Ključne besede: upravljanje odnosov s strankami, poslovni procesi, nepremičnine, eno-stranska spletna aplikacija, REST, Django, AngularJS.

Abstract

Title: Development of web application for customer relationship management in the real estate field.

Author: Rok Založnik

Nowadays in given economic situation it is crucial, if not almost necessary for companies to have good relationship with their customers. Those companies that are aware of this fact, are introducing new business strategy, which focuses on good customer relationships, using applications for customer relationship management (CRM). The majority of existing CRM systems are too general for practical use and good efficiency in more specific fields, such as real estate. These lead us into developing web application, which will better support the real estate field. We developed the application in Django REST Framework combined with PostgreSQL database on the server side and AngularJS framework for client side of the application. First part of diploma thesis is dedicated to introduction of the CRM business strategy, and to introduction of technologies and tools, which were used during the development. Second part of the work describes practical development of the web application. In this part application arhitecture, database model, implementation and functionality of each part of the application are described. In the last part the conclusions and ideas for future work are described.

Keywords: customer relationship management, business processes, real estate, single-page application, REST, Django, AngularJS.

Poglavje 1

Uvod

Dobri odnosi s strankami so v današnjih ekonomskih razmerah za podjetja zelo zaželjeni, če ne celo nujno potrebni. Stranke postajajo čedalje bolj zahtevne, zato jih je potrebno dobro poznati. Podjetja, ki se tega zavedajo, in teh je iz dneva v dan več, vpeljujejo dobro poznano strategijo poslovanja, ki se osredotoča na dobre odnose s stankami - CRM (ang. customer relationship management). Vpeljava strategije poslovanja oziroma pristopa CRM zahteva več kot le izbiro in namestitev ustrezne programske opreme, vendar je tudi ta nedvomno ključnega pomena. Za čim boljše rezultate je pomembno, da je informacijska rešitev kot celota prilagojena strankam in procesom določenega podjetja. Specifični sistemi CRM dosegajo večjo učinkovitost, uporabniki pa jih lažje in raje uporabljajo. Trenutno je na trgu mnogo programskih rešitev, ki pa so v večini preveč splošne za določena področja uporabe in se pogosto izkažejo kot preveč kompleksne ter neučinkovite za uporabo. Iz zgoraj navedenih razlogov se nam zdi, da na trgu primankuje sistemov CRM, ki bi bili dovolj prilagojeni posameznim področjem. Tukaj vidimo priložnost za izdelavo sodobnega sistema CRM v obliki spletne aplikacije, ki bo prilagojen področju nepremičnin, učinkovit, praktičen in enostaven za uporabo poleg tega pa bo prilagojen tudi za uporabo na mobilnih napravah. Mobilne naprave postajajo zaradi vse večjega števila uporabnikov vse bolj pomembne na vseh področjih. Meseca maja leta 2015 je bil internetni promet preko mobil-

nih naprav prvič večji kot preko računalnikov [1]. Tudi delo nepremičninskih agentov je v osnovi bolj mobilno, torej terensko. Skoraj nujno potrebno je, da lahko nepremičinski agenti kar na terenu vnesejo nepremičnino ali jo strankam pokažejo, zato je takšna prilagoditev skoraj obvezna. Podpora za delovanje na mobilnih napravah omogoča dostop do informacij kjerkoli in kadarkoli, kar privede do prihranka časa, krajših prodajnih ciklov, večje odzivnosti in posledično večjega zadovoljstva strank ter uporabnikov (agentov).

1.1 Namen diplomske naloge

Nepremičninski agenti se vsak dan srečujejo z velikim številom nepremičnin in strank. Za nemoteno in učinkovito delovanje morajo biti organizirani in dobro informirani, pri tem pa bi si lahko pomagali s sistemom CRM, ki bi bil prilagojen njihovim potrebam oziroma potrebam njihovih poslovnih procesov. Namen te diplomske naloge je razvoj prilagojene spletne aplikacije CRM, ki bo nepremičninskim agentom prinesla boljše orodje, s katerim bodo lažje in učinkovitejše upravljali s svojimi strankami ter posledični izboljšali poslovanje.

1.2 Struktura diplomske naloge

V prvem poglavju predstavimo namen, strukturo in cilj diplomske naloge. V drugem definiramo koncept upravljanja odnosov s strankami, v tretjem pa pregledamo in opišemo tehnologije ter orodja, ki jih uporabimo pri razvoju. Četrto poglavje vsebuje opis načrtovanja in razvoja aplikacije, prikazana pa je tudi uporaba aplikacije in njene funkcionalnosti. V zadnjem poglavju pa so podane sklepne ugotovitve dela in ideje za nadaljnje delo.

1.3 Cilj diplomske naloge

Cilj te diplomske naloge je izdelati programsko rešitev, ki bo zadovoljevala ciljno skupino uporabnikov, to je nepremičninskih agentov. Končni izdelek bo spletna aplikacija za upravljanje odnosov s strankami, prilagojena za področje nepremičnin. Razvita aplikacija mora tako:

- vsebovati funkcionalnosti, ki so specifično potrebne in koristne na področju nepremičnin,
- biti praktična in enostavna za uporabo,
- biti prilagojena za uporabo na mobilnih napravah,
- biti dobro zasnovana, tako da omogoča enostaven razvoj nadgradenj in izboljšav.

Poglavje 2

Upravljanje odnosov s strankami

V potrošniškem svetu velja, da je stranka kralj. V luči tega je zelo pomembno, da imata stranka in prodajalec oziroma ponudnik dobre odnose, kar se lahko doseže s primernim upravljanjem njunih odnosov.

2.1 Ozadje

Zelo pomembno pri vpeljavi upravljanja odnosov s strankami je, da poznamo osnovne značilnosti strategije CRM ter obstoječe rešitve.

2.1.1 Kaj je upravljanje odnosov s strankami?

Upravljanje odnosov s strankami oziroma pristop **CRM** je poslovna strategija v podjetju, ki se bolj osredotoča na izboljšanje odnosov s strankami, kot pa na same produkte. Osnovna ideja, ki stoji za to strategijo, je dobro poznavanje strank, njihovih navad in potreb, kar stranke zelo cenijo, ter se povrne v obliki dolgoročnega sodelovanja in višjega zaslužka za podjetje [2]. Takšna strategija obravnavanja strank zahteva redno komunikacijo in podporo strankam na vseh področjih. Zaposleni se vseskozi neposredno soočajo s strankami na področju podpore, prodaje in trženja. S tem tudi pridobivajo

podatke in gradijo bazo znanja, kar jim omogoča boljše poznavanje strank, njihovih potreb ter njihovega vedenja. Dobri odnosi so ključni za dolgoročno sodelovanje in za optimizacijo poslovnih procesov, kar pa je ključ do poslovnega uspeha ter povečanja dobička.
Poleg dobrega poznavanja svojih strank in njihovih potreb strategija zajema tudi prepoznavanje novih poslovnih priložnosti ter pridobivanje novih, dobičkonosnih strank.
Strategija CRM zajema številne procese, zaradi česar je za hitro in učinkovito poslovanje skoraj nujna uporaba oziroma podpora IT-ja. Zaradi vsega tega nekatere organizacije danes uporabljajo programske rešitve za upravljanje odnosov s strankami, t.i. sisteme CRM. Takšni sistemi so zasnovani tako, da nudijo vse informacije o strankah, produktih in poslih na enem mestu. Poleg tega vsebujejo še ostale podporne funkcionalnosti, od kreiranja poročil, pa do načrtovanja in pošiljanja kampanj elektronske pošte.

2.1.2 Procesi in značilnosti strategije CRM

Različna podjetja poslujejo različno in imajo specifične storitve oziroma produkte. Zaradi tega so se razvile in se še razvijajo različne vrste specifičnih sistemov CRM. Kljub temu so si v splošnem cilji in glavne funkcionalnosti teh sistemov precej podobni, če ne enaki. Glede na upravljanje in poslovanje lahko razdelimo sisteme CRM na tri podpodročja [3]:

1. Operativni sistem CRM: glavni cilj je avtomatizacija prodaje, trženja in podpore strankam.

2. Analitični sistem CRM: vloga analitičnega sistema CRM je pridobivanje podatkov o strankah iz različnih virov in analiza le teh ter poročanje, z namenom izboljšanja storitev in poslovnih procesov.

3. Sodelovalni sistem CRM: komunikacija med dobavitelji, prodajalci in strankami ter koordinacija procesov.

Kot vidimo, vsa zgornja podpodročja skupaj pokrivajo funkcionalnosti celotnega sistema CRM. Ključne značilnosti, ki jih ima nek splošen sistem CRM so torej:

- podpora prodaji,
- podpora trženju,
- podpora strankam,
- poročanje,
- sodelovanje,
- povezljivost do zunanjih sistemov.

2.1.3 Sistem CRM - informacijske rešitve

Za hitro in učinkovito uporabo strategije CRM v praksi je danes ključna dobra informacijska podpora. Z uporabo sodobnih informacijskih tehnologij (baza podatkov, internet, elektronska pošta, itd.) lahko sistemi CRM združujejo podatke iz različnih virov in uporabnikom nudijo celostni pogled na stranko v realnem času. Poleg tega lahko hranijo različne dokumente, podatke o sestankih, telefonskih klicih, opravilih in drugo. S sistemom lahko povežemo svojo elektronsko pošto in izvajamo t.i. kampanje elektronske pošte ter pošiljamo celo SMS sporočila. Generiramo lahko različne dokumente, kot so na primer poročila ali pa razni drugi dokumenti. Možnosti je veliko.
Sistem CRM je informacijska rešitev oziroma računalniški program, ki na enem mestu združuje in omogoča vse funkcionalnosti za podporo poslovanja po strategiji CRM. Je celovit poslovni informacijski sistem.

2.1.4 Obstoječe programske rešitve

Že samo v Sloveniji je kar nekaj ponudnikov, ki ponujajo sisteme CRM. Velika večina takšnih sistemov je zelo splošnih in pokrivajo praktično vsa področja

uporabe. Takšni sistemi so sicer ustrezni skoraj za vsako organizacijo, vendar pa niso nikjer idealni. Nekatere plačljive pa tudi odprtokodne rešitve sicer ponujajo možnost prilagoditve posameznih modulov in polj entitet, vendar imajo tudi te svoje omejitve.
Slabost takšnih splošnih, večnamenskih programskih rešitev je ta, da s seboj prinesejo veliko nepotrebnih funkcionalnosti, po domače "balasta", in hitro postanejo nepregledne ter neuporabne. Hitrost in učinkovitost takšne programske rešitve je omejena, še posebej v primerjavi s prilagojenimi sistemi CRM, ki poudarjajo specifičnost posameznega področja. V teoriji in praksi zasledimo osnovne vrste takšnih sistemov [4]:

- eCRM (ang. electronic CRM),
- ECRM (ang. enterprise CRM),
- PRM (ang. partners CRM),
- cCRM (ang. collaborative CRM),
- SRM (ang. supplier CRM),
- xCRM (novejše hibridne oblike CRM).

Ponudnikov sistemov CRM je na trgu iz dneva v dan več, najpomembnejšo vlogo pa tako v svetovnem merilu kot v Sloveniji igrajo tisti, ki so namenjeni srednje velikim in velikim podjetjem. V Sloveniji se najpogosteje uporabljajo plačljivi sistemi: Intrix CRM, Salesforce, Microsoft Dynamics CRM, SAP CRM, Oracle CRM in drugi.
Nekateri ponudniki ponujajo cenejše odprtokodne rešitve in prilagoditve le teh, vendar so takšne aplikacije ponavadi slabo podprte, zastarele in počasnejše v primerjavi z zgoraj omenjenimi.

2.1.5 Najpogostejši uporabniki sistemov CRM

Uporaba sistema CRM je smiselna skoraj v vsakem podjetju, ki ima stranke, toda največja dodana vrednost uporabe sistema je na posameznih področjih

industrije. Najbolj perspektivna področja, ki uporabljajo takšne sisteme so področja [5]:

- prodaje,
- poslovnih storitev,
- bančništva, zavarovalništva in financ,
- proizvodnje.

2.2 Specifični sistem CRM za popolno podporo

Ne bi pretiravali, če bi dejali, da je sistem CRM nujno potreben za podjetje, ki želi zadovoljne in zveste stranke. Zahteve strank naraščajo, saj le te želijo izboljšane storitve, zato je še kako pomembno, da podjetja svoje stranke čimbolje poznajo.
Toda tu se je potrebno zavedati, da niso vse stranke enake, prav tako niso enake storitve in produkti, ki jih ponujajo različna podjetja. Poslovanje lahko podjetja izboljšajo tako, da se čimbolj prilagodijo svojim strankam in njihovim potrebam ter poslovnim procesom, ki jih izvajajo.

2.2.1 Sistem CRM za nepremičninske agente

Eno izmed področij, kjer splošen sistem CRM ni najbolj primeren, je področje trgovanja z nepremičninami, s katerimi se ukvarjajo nepremičninske agencije in nepremičninski agenti. Njihovo delo je iskanje primernih nepremičnin za stranke, ki iščejo oziroma kupujejo določeno nepremičnino, in iskanje kupcev za tiste stranke, ki neko nepremičnino prodajajo ali oddajajo v najem.
Tu so produkti torej nepremičnine, kontakti pa so kupci oziroma morebitni kupci nepremičnin in prodajalci oziroma morebitni prodajalci nepremičnin. Potencialne stranke pa predstavljajo zasebni prodajalci, ki bi jih agenti lahko

zastopali, vendar jih še ne. Poslovni procesi, ki jih vsakodnevno uporabljajo nepremičninski agenti, so torej precej jasni, vendar specifični.

Poglavje 3

Pregled in opis uporabljenih tehnologij ter orodij

V tem poglavju bomo pregledali in opisali orodja in tehnologije, ki smo jih uporabili pri razvoju spletne aplikacije. Pri tem se ne bomo spuščali v podrobnosti vseh tehnologij, saj to ni namen tega dela, zato bomo nekatere samo omenili. Bolj kot sama orodja in tehnologije je pomembna uporaba le teh. Več pozornosti bomo namenili načrtovanju in razvoju sistema CRM, arhitekturi, funkcionalnostim ter dobri uporabniški izkušnji.

Preden nadaljujemo z opisom uporabljenih tehnologij bi radi izpostavili, da se naša aplikacije razlikuje od večine klasičnih aplikacij in spletnih strani, ki jih poznamo. Pri nas je celotna programska rešitev sestavljena iz dveh delov. Prvi del je strežniški del, ki pri običajnih aplikacijah zgradi vsako zahtevano stran in jo pošlje uporabniku, pri nas pa ta del zgradi le prvo stran, od tu naprej pa strežnik zagotavlja le še podatke, gradnja strani pa se zgodi na drugem delu, to je na strani odjemalca. Takšno vrsto aplikacije imenujemo SPA (ang. single-page application), več o tem pa bomo povedali v nadaljevanju.

Zaenkrat je pomembno povedati, da smo strežniški del implementirali v programskem jeziku Python, z uporabo ogrodja Django REST Framework, odjemalski del pa z uporabo JavaScript ogrodja AngularJS.

3.1 Git

Git je sistem za nadzor verzij oziroma VCS (ang. version control system) in se ga zelo pogosto uporablja pri razvoju aplikacij. Omogoča, da več uporabnikov dela istočasno na istem projektu. Pomembna prednost uporabe je nadzor in upravljanje verzij programa. Mi smo ga pri razvoju uporabili predvsem zaradi preprečitve neželenih sprememb ali celo morebitne izgube podatkov.

3.2 Django REST Framework in Django

Django Rest Framework (DRF) je ogrodje za izdelavo končnih točk API (ang. application programming interface), ki temelji na ogrodju Django. DRF že vsebuje podporo za avtentikacijo, lažje upravljanje dovoljenj in pravic, upravljanje različic, in drugih funkcionalnosti, ki jih potrebujemo za hitro in enostavno vzpostavitev spletnih storitev temelječih na arhitekturi REST [6].

Django pa je odprtokodno MVC (ang. model-view-controller) ogrodje za izdelavo spletnih aplikacij, napisano v programskem jeziku Python. Prva različica je bila javno izdana 21. julija 2005, ogrodje pa je vzdrževano pod okriljem neprofitne organizacije Django Software Foundation (DSF) [7].
Glavna prednost Djanga je hitra in enostavna izdelava kompleksnih spletnih aplikacij, ki uporabljajo podatkovne baze. Filozofija ogrodja temelji na večkratni uporabi komponent, hitrem razvoju in principu DRY (ang. don't repeat yourself). Ogrodje ima vgrajen modul ORM (ang. object-relational mapping). To je mediator med podatkovnim modelom (razred v Pythonu) in podatkovno bazo, ki nam olajša povezovanje ter komuniciranje z bazo.

3.3 PostgreSQL

PostgreSQL je zmogljiv, odprtokoden sistem za upravljanje s podatkovnimi bazami oziroma ORDBMS (ang. object-relational database management system). Prva različica je bila izdana že leta 1996 [8]. Glavna funkcija je seveda

varna hramba podatkov, ki pa je zanesljiva in hitra. Tekmuje s številnimi konkurenti, vendar je za razliko od nekaterih drugih (npr. MSSQL, Oracle, itd.) tudi brezplačen.

3.4 AngularJS

AngularJS je odprtokodno JavaScript MVC ogrodje za izdelavo aplikacij na strani odjemalca. Prva različica je bila izdana konec oktobra 2010, s strani Googla, ki ga skupaj s skupino posameznikov podpira in še naprej razvija. Trenutno (polovica leta 2016) je v fazi razvoja Angular 2 (beta verzija), ki prinaša velike sintaktične in konceptualne spremembe [9].

Glavna filozofija tega priljubljenega ogrodja je ločitev manipulacije DOM (ang. document-object model) elementov (drugače povedano upravljanje gradnikov spletnih strani), ki se izvaja na odjemalčevi strani, od poslovne logike, ki se izvaja na strežniški strani. To med drugim omogoča sočasni razvoj in ponovno uporabo posameznih komponent na obeh straneh.
AngularJS implementira MVC arhitekturni vzorec aplikacij, ki ločuje aplikacijo v različne dele, vsakega s svojo odgovornostjo. V tem kontekstu model (ang. model) definira podatke za prikaz, pogled (ang. view) pa skrbi za prikaz podatkov iz modela in omogoča dvosmerno spreminjanje podatkov (ang. 2-way data binding). To pomeni, da kadarkoli se spremeni pogled, se spremeni tudi model. Nadzornik (ang. controller) vsebuje kodo, ki se odziva na akcije uporabnika in kodo za pripravo podatkov modela.

Ogrodje vsebuje številne komponente, s katerimi lahko gradimo sodobne, "bogate" internetne aplikacije. Spodaj bomo omenili le nekaj pomembnih izrazov in komponent tega ogrodja:

- **Izraz (ang. expression)**: omogoča dostop do spremenljivk in funkcij znotraj predlog.

- **Kontekst (ang. scope)**: kontekst, ki povezuje objekte z določenim modelom, deluje kot lepilo med modelom in pogledom.
- **Nadzornik (ang. controller)**: vsebuje poslovno logiko za prikaze.
- **Storitev (ang. service)**: vsebuje poslovno logiko neodvisno od prikaza.
- **Direktiva (ang. directive)**: je komponenta, ki predstavlja nov HTML element. Omogoča dostop do spremenljivk in funkcij naše aplikacije znotraj HTML-ja.
- **Modul (ang. module)**: del aplikacije, ki je lahko nadzornik, storitev ali katera druga komponenta.
- **Predloga (ang. template)**: HTML datoteka, ki vsebuje tudi direktive. Skrbi za prikaz podatkov iz nadzornika in modela.
- **Filter**: formatira izpis za ustrezen prikaz.

3.5 Grunt

Grunt je orodje za opravljanje ponavljajočih se opravil oziroma avtomatizacijo. Na kratko povedano nam Grunt olajša izvajanje opravil v JavaScriptu. Najpogosteje se ga uporablja za: minifikacijo datotek, kopiranje datotek, testiranje in druga opravila.

3.6 Bootstrap

Bootstrap je brezplačno odprtokodno ogrodje za izdelavo odzivnih (ang. responsive) spletnih strani in aplikacij. Odzivne spletne strani so takšne, kjer se izgled strani prilagaja velikosti okna (brskalnika). S tem lahko dosežemo, da je stran prilagojena tudi za mobilne naprave, kot so telefoni in tablice. Vsebuje HTML in CSS predloge za ključne elemente vsake strani (npr. obrazci,

gumbi, navigacija, itd.) ter z njimi povezane JavaScript razširitve. Prva verzija tega ogrodja je bila izdana sredi leta 2011, in sicer s strani Twitterja [10]. Trenutno je v izdelavi četrta verzija Bootstrap ogrodja.

3.7 HTML5 in CSS3

HTML5 je označevalni jezik, ki se uporablja za definiranje dokumentov na spletu. Gre za peto, trenutno, verzijo HTML standarda, ki je bila objavljena konec leta 2014 [11]. Jezik sestavljajo značke, atributi in vsebina značk. S temi elementi definiramo postavitev vsebine (besedilo, slike, videoposnetki, itd.) na strani.

Poleg HTML-ja se pri oblikovanju strani uporablja še kaskadne slogovne pole oziroma CSS (ang. cascading style sheets). CSS pa je jezik, s katerim natančneje oblikujemo stran. S CSS selektorji lahko izberemo več ali pa posamezen HTML element znotraj neke spletne strani. Tako izbranim elementom lahko spreminjamo lastnosti. Določimo lahko vse od pisave, barv, razmika, itn. CSS3 je zadnja različica in je izšla leta 1999.

Poglavje 4

Razvoj aplikacije

Pred samim razvojem aplikacije moramo natančno analizirati zahteve uporabnikov in na podlagi teh določiti osnovne funkcionalnosti programa. Ko enkrat vemo, katerim zahtevam moramo zadostiti in kaj bomo razvijali, lahko izberemo platformo za razvoj. V našem primeru je pomembna tudi dobra uporabniška izkušnja, zato smo se odločili, da bomo izdelali t.i. eno-stransko aplikacijo oziroma SPA. Glavna značilnost eno-stranskih aplikacij je, da so v številnih lastnostih podobne klasičnim namiznim aplikacijam. Za takšno aplikacijo je potrebno veliko JavaScript kode in manipulacije strani, zato smo se odločili za dvo-delno arhitekturo, sestavljeno iz strežniškega dela, ki zagotavlja storitve, in odjemalskega dela, ki skrbi za prikaz ter vnos podatkov. Za lažjo implementacijo dodelanih uporabniških vmesnikov smo uporabili ogrodje AngularJS. Drugi pomemben cilj, ki smo mu sledili, je uporabnost aplikacije na mobilnih napravah. Zato smo aplikacijo naredili odzivno in prilagodljivo, pri čemer smo si pomagali z ogrodjem Bootstrap. V tem poglavju si bomo na hitro pogledali arhitekturo aplikacije in implementacijo posameznih sklopov. Pri odjemalskem delu bomo sproti razložili tudi funkcionalnosti in primere uporabe.

4.1 Arhitektura aplikacije

Naša aplikacija je sestavljena iz dveh delov. Prvi, strežniški del je implementiran v ogrodju Django in teče na spletnem strežniku (npr. Apache, Nginx, Cherokee, itd.). Ta del nam omogoča upravljanje s podatki, ki jih hranimo v PostgreSQL podatkovni bazi. Do podatkov lahko dostopamo, jih urejamo, dodajamo ali brišemo preko končnih točk API, ki smo jih definirali. Poleg tega skrbi strežniški del tudi za integriteto podatkov in varnost, lahko pa izvaja tudi razne procedure ter skrbi za višje-nivojsko poslovno logiko, ki mora biti enaka za vse odjemalce. Takšna arhitektura nam zlahka omogoča tudi izdelavo "prave" mobilne aplikacije, saj bi ta do podatkov lahko dostopala preko istih spletnih storitev, pri čemer bi bilo potrebno izdelati le nov odjemalski del.

Drugi, vidnejši del naše aplikacije pa predstavlja odjemalski del, ki je kot pri večini spletnih aplikacij, kar uporabniški vmesnik. Tega smo implementirali v tehnologijah JavaScript, HTML5 in CSS3, pri tem pa smo uporabili ogrodji AngularJS in Bootstrap. S pomočjo ogrodja AngularJS smo v jeziku JavaScript spisali logiko, ki se izvaja na odjemalcu in implementirali interaktivne strani, tako da smo razširili HTML elemente z lastnimi direktivami. Bootstrap smo uporabili za izdelavo lepših, odzivnih strani. S pomočjo obeh ogrodij smo zagotovili bolj tekočo uporabniško izkušnjo in lepše ter odzivnejše uporabniške vmesnike.
V odjemalskem delu podatkov ne shranjujemo, razen za potrebe avtentikacije, kjer uporabljamo lokalno shrambo v odjemalčevem brskalniku. Aplikacija uporablja prej omenjene spletne storitve, preko katerih pridobiva oziroma shranjuje podatke. V našem primeru končne točke API sledijo RESTful arhitekturnem slogu. Slog mora zadoščati naslednjim arhitekturnim omejitvam [12]:

- ločitev odjemalca in strežnika,
- brez stanja (ang. stateless), komunikacija mora vsebovati vso potrebno

informacijo,

- uporaba medpomnenja (ang. caching) za ponovno uporabo odgovorov v kasnejših zahtevah,
- večplastni sistem (ang. layered system), odjemalec se ne zaveda neposredno s kom komunicira,
- enoten vmesnik (ang. uniform interface), vsi odjemalci dostopajo do storitev po enotnem vzorcu.

Ključne lastnosti RESTful arhitekturnega sloga so:

- skalabilnost,
- splošnost,
- neodvisnost,
- odzivnost na spremembe,
- latenca,
- varnost,
- enkapsulacija.

Pri tem se uporablja protokol HTTP, podatki pa se prenašajo v formatu JSON.

4.1.1 Eno-stranska aplikacija

Našo aplikacijo lahko opredelimo tudi kot eno-stransko aplikacijo oziroma SPA (ang. single-page application), ker je v številnih lastnostih podobna namiznim aplikacijam.
Pri klasičnih spletnih aplikacijah se celotna stran zgradi na strežniku in nato pošlje odjemalcu, ki jo prikaže. Enako se zgodi za vsako posamezno stran,

ki jo zahteva odjemalec. Pri aplikaciji SPA je zgodba drugačna. V primeru aplikacij SPA se samo prva stran zgradi na strežniku, nato pa se za vsako nadaljnjo stran, ki jo zahteva odjemalec, pošljejo samo še podatki (Slika 4.1). Zaradi tega aplikacija, potem ko je enkrat naložena, deluje hitreje in bolj tekoče. Od tod tudi ime eno-stranska, saj se samo prvič prenese stran, potem pa samo še deli strani oziroma podatki.

Slika 4.1: Arhitektura aplikacije SPA

Glavni prednosti aplikacij SPA sta bolj tekoča uporabniška izkušnja in manjša obremenitev strežnika, saj ta ni več zadolžen za gradnjo strani ter pošilja le še gole podatke [13].

4.2 Podatkovni model

Po postavljenih osnovnih zahtevah, smo pričeli z razvojem aplikacije. Najprej smo se lotili strežniškega dela in pripadajoče podatkovne baze. V nasprotju s tradicionalnim razvojem, kjer se celotni podatkovni model razvije na začetku, smo uporabili agilnejši pristop. Podatkovni model smo razvijali sproti, v skladu z modulom, ki smo ga v tistem času razvijali.

Entitetno-relacijskega modela nismo razvili v načrtovalskem programu kot je MySQL Workbench ali kaj podobnega, temveč smo pisali razrede v Pythonu. Shemo podatkov smo definirali v datotekah *models.py*, znotraj vsakega Django "app-a". V Djangu je "app" sklop povezanih funkcionalnosti aplikacije, neke vrste modul. Razširitev razreda *models.Model* v okolju Django predstavlja tabelo, lastnosti razreda pa definirajo posamezne atribute in entitetne tipe podatkovnega modela. Z izvedbo dveh preprostih ukazov (Koda 4.1) Djangov vgrajeni modul ORM sam sinhronizira našo podatkovno shemo v datotekah *models.py* s podatkovno bazo in v njej ustvari dejanske tabele [14]. Ta funkcionalnost nam olajša delo in privarčuje veliko časa.

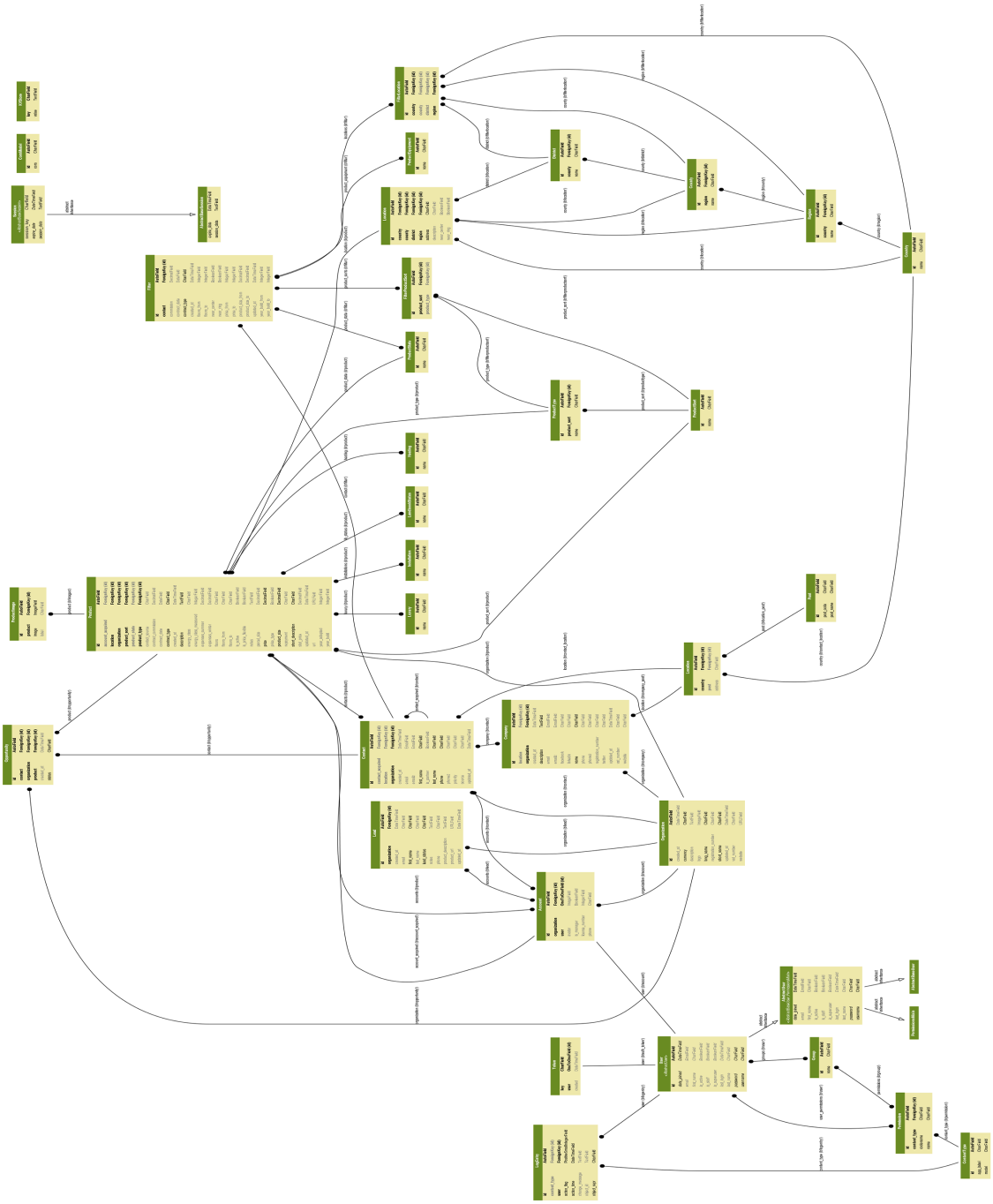
Koda 4.1: Migriranje podatkovne sheme v podatkovni bazi.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Podatkovni model smo sproti med razvojem ustrezno normalizirali, česar v nadaljevanju ne bomo več posebej poudarjali. Kot primer normalizacije lahko navedemo lokacijo pri entiteti *Product* (opisan v naslednjem podpoglavju), ki je sestavljena iz večih atributov (država, regija, občina, četrt). Za te atribute smo znotraj sklopa ustvarili entiteto *Location*, entiteto *Product* pa smo s tujim ključem povezali z njo. Podobno smo storili tudi za ostale primerljive primere.

4.2.1 Struktura podatkovnega modela

Pri večjih projektih programska koda relativno hitro postane nepregledna in začne prihajati do zapletov ter podvajanj vsebine. Temu v izogib smo strežniški del aplikacije in podatkovni model zgradili modularno. Projekt smo razdelili na več vsebinsko povezanih modulov oziroma tako imenovanih "app-ov". Znotraj vsakega smo definirali pripadajoči del podatkovnega modela, zato bomo podatkovni model tako tudi predstavili. Slika 4.2 prikazuje podatkovni model, ki smo ga razvili za potrebe naše aplikacije. Več o sami strukturi strežniškega dela bomo izvedeli v naslednjem podpoglavju.



Slika 4.2: Podatkovni model

4.2.1.1 Uporabniki in organizacija

Uporabniki naše aplikacije so nepremičninski agenti. Izvirni model uporabnika, ki pride zraven Djanga, hrani osnovne podatke o uporabniku: uporabniško ime, geslo, elektronska pošta, ime in priimek.
Za potrebe naše aplikacije smo ta model razširili z entiteto *Account* (Koda 4.2), katere atributi so: telefon (*telephone*), licenca agenta (*license_number*), slika uporabnika (*avatar*), logična vrednost (*is_manager*), ki nam pove ali je agent upravljalec ali ne, jezik (*language*) in tuji ključ *organization*, ki agenta povezuje z organizacijo.

Koda 4.2: Izvorna koda entitete Account

```
class Account(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    phone = models.CharField(max_length=15, blank=True)
    license_number = models.IntegerField(blank=True, null=True)
    avatar = ImageField(blank=True, null=True, default='images/
        user-default.png', upload_to=upload_to)
    language = models.CharField(choices=settings.LANGUAGES,
        default='en', max_length=4)
    is_manager = models.BooleanField(default=False)
    organization = models.ForeignKey(Organization, null=True)

    def __str__(self):
        return ("%s %s") % (self.user.first_name, self.user.
            last_name)
```

V tem sklopu smo definirali tudi entiteto *Organization*, saj želimo, da naša aplikacija podpira več organizacij, torej več nepremičninskih agencij hkrati. Entiteta hrani podatke o organizaciji: kratek naziv in dolg naziv podjetja, opis dejavnosti, davčna in matična številka, logotip ter spletna stran.
Vse pomembne entitete v nadaljevanju smo povezali z organizacijo, kar nam omogoča, da lahko uporabnik vidi le podatke znotraj svoje organizacije, mi pa lahko za več organizacij uporabimo le eno podatkovno bazo. To nam omogoča lažje vzdrževanje, pregled in nižje stroške.

4.2.1.2 Produkti

Najbolj specifičen del naše aplikacije CRM so nepremičnine. Nepremičnino smo v kontekstu strategije CRM poimenovali *Product* in predstavlja tudi največjo entiteto našega modela. Ta vsebuje številne podatke o nepremičnini in še nekaj drugih podatkov, pomembnih za agente: zapiski agenta, povezava z agenti, podatek o tem, kateri agent je pridobil nepremičnino, vir pridobitve kontakta, provizija agenta in veljavnost pogodbe.
Podatki o nepremičnini so: vrsta pogodbe (prodaja oziroma oddaja), vrsta nepremičnine, lokacija nepremičnine, cena nepremičnine, vrsta cene (enota ali kvadratni meter), leto izgradnje, leto adaptacije, površina, površina parcele, kratek in podroben opis, stanje nepremičnine, stanje opremljenosti, energijski razred, stanje zemljiške knjige, opremljenost, priključki in še nekaj drugih.
Produkt je povezan z enim ali večimi kontakti (*Contact*). V tem primeru so povezani kontakti prodajalci nepremičnine, več o njih bomo povedali v nadaljevanju tega podpoglavja.

Znotraj sklopa smo definirali tudi entitete, ki predstavljajo šifrante za opis lokacije (država, regija, občina, četrt) in šifrante za ostale izbirne lastnosti nepremičnine (npr. stanje nepremičnine, energijski razred, priključki, itd). Za slike nepremičnin smo definirali entiteto *ProductImage*.

4.2.1.3 Kontakti

Strategija CRM se osredotoča na stranke, te torej predstavljajo najpomembnejši del aplikacije. Stranke bomo v naši aplikaciji imenovali kontakti. To so fizične osebe, ki so lahko tudi kupci ali prodajalci oziroma oboje hkrati.
Entiteta *Contact* hrani osnovne kontaktne podatke kot so: ime, priimek, elektronska pošta, telefon, lokacija, podjetje in še nekaj drugih. Povezana je tudi z organizacijo (*Organization*) in agenti (*Account*).

Znotraj sklopa smo definirali tudi entiteto *Filter*. Ta se uporablja za hranjenje podatkov o nepremičnini, katero išče oziroma kupuje stranka (kontakt).

Gre torej za iskalni filter, preko katerega v nadaljevanju poiščemo ujemanje kontakta in nepremičnine. Takemu ujemanju rečemo priložnost, o tem bomo več povedali v nadaljevanju. Filter je torej povezan s kontaktom, na drugi strani pa vsebuje številne atritbute, podobne tistim v entiteti *Product*: vrsta pogodbe, lokacije nepremičnine, cena od in cena do, stanje nepremičnine ter še nekatere.

4.2.1.4 Potencialne stranke

Še en pomemben del strategije CRM predstavljajo potencialne stranke oziroma *Leads*. Tukaj smo definirali entiteto *Lead*, katera je zelo podobna kontaktu in vsebuje osnovne kontaktne podatke neke osebe. Ravno tako je povezana z agenti in organizacijo. Kadar neka potencialna stranka postane stranka, potem to entiteto pretvorimo v kontakt. Ker pa bo v dejanski uporabi potencialnih strank verjetno precej več kot pravih, jih ne želimo mešati s pravimi kontakti, kar je razlog za uvedbo svoje entitete.

4.2.1.5 Priložnosti

Priložnost oziroma *Opportunity* je entiteta, ki hrani podatke o kontaktu in nepremičnini (produktu), ki se ujema s kontaktom, glede na kriterije njegovega iskalnega filtra. Entiteta poleg produkta in kontakta hrani tudi *status*, ki pove, ali je kontakt ujemajoči se produkt že pregledal ter ali je zanj zainteresiran ali ne. Zraven hrani še atribut ponujena cena (*offered_price*) in datum ponudbe (*date_offered*). Vsi trije podatki so zanimivi za agente, da vedo, koliko je posamezen kontakt za nepremičnino največ ponudil, in kdaj.

4.2.1.6 Podjetja

Podjetje oziroma *Company* je entiteta, ki hrani osnovne podatke nekega podjetja. Te so: naziv, opis, davčna in matična številka, spletna stran in še nekaj drugih. Kontakti so s podjetji povezani.

4.3 Strežniški del aplikacije

Nekaj o strežniškem delu aplikacije smo razložili že v uvodu. Povedali smo, da je napisan v programskem jeziku Python, realiziran pa je s pomočjo Django REST Framework ogrodja, namenjenega izdelavi REST API-jev. Njegova naloga je upravljanje s podatki, arhitektura pa sledi RESTful arhitekturnem slogu. Tudi tu smo se razvoja lotili modularno in smo sledili arhitekturi MVC in principu DRY. Za uporabo Djanga smo se odločili, ker omogoča hiter razvoj kompleksnih aplikacij, ki so skalabilne in varne. V kombinaciji z DRF je primerno orodje za izdelavo API-jev. Nekaj izkušenj s Pythonom in ogrodjem smo že imeli, kar nam je delo malce olajšalo.

4.3.1 Struktura DRF projekta

Projekt je sestavljen iz več vsebinsko in funkcionalno povezanih Python modulov oziroma "app-ov". Eden ima vedno ime, ki je enako imenu projekta. V tem hranimo nastavitve in povezave vezane na celoten projekt. Znotraj posameznega modula najdemo več datotek. Za nas so pomembne naslednje:

Datoteka models.py definira entitete podatkovnega modela. Poseben opis ni potreben, saj smo o tem govorili že v prejšnjem podpoglavju.

Datoteka views.py hrani poglede. Pogled je funkcija, ki vrača HTTP odgovor. V splošnem je odgovornost pogleda, da glede na podane parametre pridobi in vrne podatke. V DRF pogled predstavlja končno točko API (ang. endpoint), ki vrne odgovor v formatu JSON. Pogledov imamo več vrst: pogledi ki vrnejo seznam podatkov, pogledi ki vrnejo točno določen objekt, pogledi ki omogočajo vnos podatkov, pogledi ki omogočajo posodabljanje podatkov in pogledi, ki omogočajo odstranjevanje.

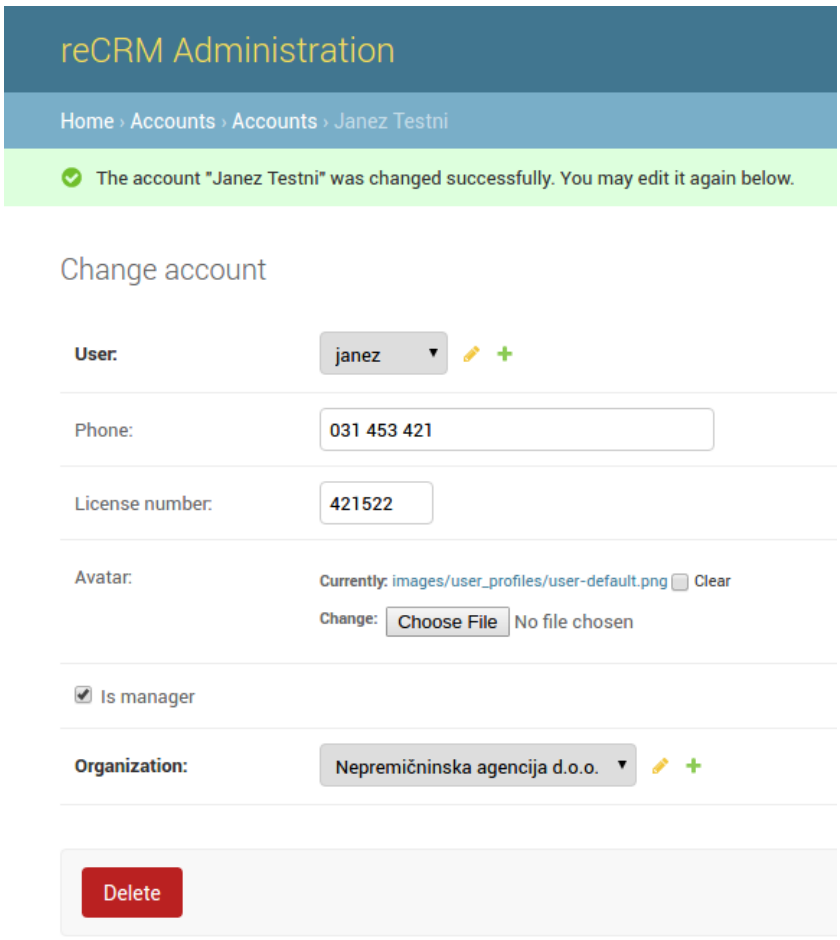
Datoteka serializers.py hrani tako imenovane pretvornike, ki skrbijo za predstavitev naših podatkov in deluje kot nekakšen most, saj pretvarja podatke iz formata JSON v objekte razumljive Djangu, in obratno.

Datoteka urls.py hrani poti, preko katerih dostopamo do naših storitev. Vsak pogled povežemo z določenim URL-jem in tako definiramo končno točko API. Obstaja pa tudi globalna datoteka *urls.py*, v kateri so definirane poti do "app-ov", kateri prevzamejo nadaljnje usmerjanje. Ta se nahaja v "app-u", ki nosi isto ime kot projekt.

Datoteka admin.py pove Djangu, katere entitete podatkovnega modela lahko skrbnik vidi in ureja na strani za skrbnike. Poleg tega, lahko v datoteki prilagodimo izgled in postavitev nekaterih gradnikov.

4.3.2 Stran za skrbnike

Ena izmed bolj uporabnih komponent Djanga je avtomatiziran uporabniški vmesnik za skrbnike strani (administratorje). Ta s pomočjo meta podatkov podatkovnega modela in datotek *admin.py* zagotavlja vmesnik, kjer lahko skrbnik ureja vsebino strani (Slika 4.3). Stran je še posebno uporabna za hitro urejanje podatkov, za katere drugače ni uporabniškega vmesnika. Tako za pregled ali vnos nekaterih podatkov skrbniku ni potrebno neposredno brskati po podatkovni bazi.
Mi ga med drugim uporabljamo za kreiranje nove organizacije in vsaj enega uporabnika, saj registracije v tej verziji aplikacije nismo implementirali.



Slika 4.3: Urejanje profila agenta na strani za skrbnike

4.3.3 Verzioniranje APIja

Dodajanje novih funkcionalnosti in odpravljanje hroščev nas silita v izdajo novih verzij spletnih storitev. Ker želimo ta proces čimbolj poenostaviti, smo se držali dobre prakse načrtovanja storitev. S tem namenom, smo vse strežniške poti definirali v novi datoteki *urls_v1.py*, v datoteki *urls.py* (Koda 4.3) pa smo ustvarili povezavo do te. Tako ob izdaji nove verzije samo naredimo povezavo do nove datoteke (npr. *urls_v2.py*). S tem omogočimo lažji prehod na novo verzijo storitev in hkrati zagotovimo, da stare verzije še vedno delujejo določen čas za lažji prehod vseh odjemalcev.

Koda 4.3: Vsebina datoteke *urls.py*

```
urlpatterns = [
    url(r'^v1/', include('recrm_api.urls_v1', namespace='v1')),
    url(r'^admin/', admin.site.urls),
]
```

4.3.4 Lokalizacija

Celotno aplikacijo smo zasnovali v angleškem jeziku, vendar želimo kljub temu podpreti tudi druge jezike. Za takšno prilagoditev moramo poskrbeti za ustrezne prevode v obeh delih aplikacije. Večina besedil, ki jih želimo prevesti se nahaja na uporabniškem vmesniku, na strežniški strani je potrebno poskrbeti le za prevode nekaterih šifrantov.
Tiste vrednosti izbirnih polj, ki smo jih definirali v kodi, smo označili za prevod s posebno funkcijo *ugettext()*, ki je del ogrodja Django. Te označene šifrante lahko kasneje prevedemo v datotekah s končnico *.po*, ki jih ustvari Django. Drugi šifranti so shranjeni v podatkovni bazi. Za prevod teh smo uporabili poseben "app" *Modeltranslation*. Ta v podatkovni bazi ustvari dodatna polja za prevedene vrednosti, ki jih lahko urejamo na strani za skrbnike.

Ob klicu storitve lahko nastavimo polje *Accept-Language* v glavi HTTP zahtevka in s tem Djangu povemo, v katerem jeziku naj vrne rezultate. Če tega

ne storimo, uporabi privzeti jezik, ki je nastavljen v datoteki *settings.py*. To je angleščina.

4.3.5 Končne točke API

Vsaka končna točka API, ki je del neke spletne storitve, je sestavljena iz pogleda, definicije poti in komponente, ki skrbi za serializacijo. Pogled je funkcija, v katerem se nahaja poslovna logika storitve. Komponenta za serializacijo je v našem primeru odgovorna za pretvorbo Pythonovih objektov v format JSON in obratno (deserializacija). Za vsako končno točko API je potrebno za pogled definirati tudi naslov, na katerem je storitev dostopna. Implementacijo API-jev nam je olajšala uporaba pogledov *ModelViewSet*, ki so že implementirani v okviru DRF-ja. Za uporabo pogleda je dovolj, da podamo model in komponento za serializacijo, za ostalo pa poskrbi DRF. Ta zbirka pogledov v enem zagotavlja storitev za vse osnovne akcije nekega modela: ustvari, preberi, posodobi, izbriši (CRUD - create, read, update, delete) in seznam. Delo nam olajša tudi pri definiciji poti, saj ni potrebno ročno povezati vseh akcij. Pomagamo si s funkcijo *register*, s katero definiramo povezavo do pogleda, funkcija pa sama registrira poti do posameznih akcij (Koda 4.4).

Koda 4.4: Primer registracije poti za zbirko pogledov *urls.py*

```
from rest_framework.routers import SimpleRouter

router = SimpleRouter(trailing_slash=False)
router.register(r'accounts', views.AccountViewSet)
```

Uporabniki in organizacija (accounts) je sklop, ki vsebuje API-je za upravljanje podatkovnih modelov uporabnika, uporabniškega profila in organizacije. Vsebuje tudi storitev za avtentikacijo in menjavo gesla uporabnika.

Produkti (products) predstavljajo najobširnejši sklop aplikacije, saj ima entiteta veliko atributov in povezanih šifrantov. Glavna storitev je do-

stopna na naslovu `/api/v1/products/` in nam v sklopu zbirk pogledov omogoča vse osnovne funkcije nad objekti.

Poleg tega smo v tem "app-u" implementirali še dve končni točki API. Prva, dostopna na naslovu `/api/v1/products/metadata`, nam vrača seznam vseh šifrantov, ki jih odjemalska aplikacija potrebuje za napolnitev izbirnih seznamov pri dodajanju oziroma urejanju nepremičnine. Druga pa nam, podobno kot prva, vrača seznam nepremičnin, vendar le s podatki, ki jih v odjemalski aplikaciji na seznamu nepremičnin prikazujemo. Ta storitev ni nujno potrebna, je pa smiselna za optimizacijo. Dostopna je na poti `/api/v1/products/list`.

Kontakti (contacts) so sestavljeni podobno kot sklop produkti. Glavna storitev omogoča manipulacijo entitete *Contact*, zraven pa imamo še pogleda, ki vračata seznam meta podatkov, in okrnjen seznam kontaktov. Pot storitve je `/api/v1/contacts/`.

Potencialne stranke (leads) se ne razlikujejo veliko od prejšnjih dveh. Vsebujejo enake storitve, dostopne pa so na naslovu `/api/v1/leads`.

Priložnosti (opportunities) so v kontekstu strategije CRM, ko se nek aktiven produkt (nepremičnina) ujema z iskalnimi kriteriji nekega kupca. Vsaka priložnost ima tudi status in nekaj drugih meta podatkov, zato hranimo priložnosti v podatkovni bazi, kot svojo entiteto.

Primerjava vseh produktov z vsemi iskalnimi filtri vseh kupcev je relativno kompleksna, zato bi sčasoma lahko prišlo do počasnega delovanja, če bi vse možne priložnosti preverjali znova, ob vsakem klicu storitve. Ker tega ne želimo, smo iskanje optimizirali tako, da vsakič, ko nekdo doda, uredi ali zbriše nek produkt ali kontakt, preverimo, če se pojavi kakšna nova priložnost oziroma kakšna priložnost zbledi. S takšnim pristopom pregled vseh možnih kombinacij ni več potreben, kar je precej učinkovitejše in hitrejše. Za lažje iskanje priložnosti smo spisali funkcijo *is_opportunity(filter, product)*, ki nam za podani iskalni filter kupca in podan produkt vrne logično vrednost *True* ali *False* (Koda 4.5).

Storitev, ki vrača seznam priložnosti je dostopna na naslovu `/api/v1/opportunities/`. Ob klicu brez podanih parametrov storitev vrne vse priložnosti, ob podanem parametru *contact_id* vrne priložnosti za kontakt in podobno za podan parameter *product_id*. Ta dva API-ja v odjemalskem delu aplikacije uporabimo za pridobitev podatkov o priložnostih v podrobnem pregledu produkta ali kontakta.

Koda 4.5: Funkcija is_opportunity, ki preveri, če je obstaja kakšna nova priložnost

```
'''
    Compare filter to product, return true if opportunity
    else false
'''
def is_opportunity(f, p):
    opportunity = True

    # Contract type - required
    if f.contract_type == '3' or f.contract_type == p.
        contract_type:
        pass
    else:
        return False

    # p sort and type - required
    product_sort_matches = False
    for fst in f.product_sorts.    ():
        if ((p.product_sort == fst.product_sort and fst.
            product_type == None) or
            (p.product_sort == fst.product_sort and p.
                product_type == fst.product_type)):
            product_sort_matches = True
            break
    if product_sort_matches:
        pass
    else:
```

```
        return False

    # p location - required
    location_matches = False
    for l in f.locations.    ():
        if ((l.country == p.location.country and l.region ==
             p.location.region and l.county == None
              and l.district == None)
             or (l.country == p.location.country and l.region
                  == p.location.region and
                          l.county == p.location.county and l.
                              district == None)
             or (l.country == p.location.country and l.region
                  == p.location.region and
                          l.county == p.location.county and l.
                              district == p.location.district)
                              ):
             location_matches = True
             break

    if location_matches:
        pass
    else:
        return False

    if (f.near_center == True and p.location.near_center ==
        True) or f.near_center == False:
        pass
    else:
        return False

    if (f.near_ring == True and p.location.near_ring == True
        ) or f.near_ring == False:
        pass
    else:
        return False

    # Price
```

```
    if ((f.price_from and f.price_to and f.price_from <= p.
        price <= f.price_to)
            or (f.price_from and not f.price_to and f.
                price_from <= p.price)
            or (not f.price_from and f.price_to and p.price
                <= f.price_to)
            or (not f.price_from and not f.price_to)):
        pass
    else:
        return False

    # ...
    return opportunity
```

Skupno (common) je "app", ki smo ga ustvarili za poizvedbe, ki se ne navezujejo na posamezen modul, ampak so skupne. Uporabili smo ga za implementacijo API-ja, ki vrača uporabne statistične podatke za nadzorno ploščo. Poleg tega smo tu implementirali tudi storitev za kreiranje Word dokumentov.

4.4 Odjemalski del aplikacije

Razvoja odjemalnega dela aplikacije smo se lotili s pripravo ustrezne strukture projekta AngularJS. Poleg ogrodja AngularJS, smo za izgled spletnega vmesnika uporabili plačljivo tematsko predlogo, ki temelji na Bootstrapu. Struktura in zasnova aplikacije sta nam bili pri razvoju pomembni, saj smo uporabili veliko različnih knjižnic JavaScript in CSS, kar lahko ob slabi zasnovi aplikacije povzroči počasno nalaganje in delovanje aplikacije. Da bi se izognili omenjenim problemom, smo si pomagali z orodjem Grunt. Ta nam je pomagal avtomatizirati opravila. S tem orodjem, smo na koncu tudi enostavno minimizirali nekatere datoteke in pripravili projekt za produkcijo. Dodatne knjižnice, s katerimi smo ustvarili bogat uporabniški vmesnik, pa smo tekom razvoja naložili s pomočjo orodja npm, ki je upravljalnik paketov za JavaScript.

4.4.1 Struktura AngularJS projekta

Podobno kot pri strežniškem delu smo tudi tu AngularJS aplikacijo razdelili v več vsebinsko povezanih modulov. Modul je v okolju AngularJS zbiralnik za različne gradnike aplikacije (nadzorniki, direktive, storitve, itd.). Znotraj vsakega modula imamo vsaj datoteki *controllers.js* in *services.js* ter mapo *views*. V prvi datoteki so nadzorniki (ang. controllers), v drugi pa storitve za omenjeni modul. Znotraj mape *views* se nahajajo datoteke HTML, ki predstavljajo delne poglede (ang. partials). Z njihovo pomočjo prikazujemo podatke oziroma obrazce za vnos podatkov.
V korenski mapi projekta najdemo datoteke: *app.js*, *config.js* in *config.lazyload.js*. V datoteki *app.js* so definirani vsi moduli aplikacije in njihove odvisnosti ter globalne spremenljivke [15] (Koda 4.6).

Koda 4.6: Definicija modulov - datoteka *app.js*

```
// Declare modules
angular.module('crmApp.authentication', []);
angular.module('crmApp.account', []);
angular.module('crmApp.product', []);
angular.module('crmApp.contact', []);
angular.module('crmApp.lead', []);
angular.module('crmApp.opportunity', []);

angular.module('crmApp', [
        'ui.router', 'ui.utils', 'angular.filter', 'oc.lazyLoad'
            , 'gettext', 'ngStorage', 'ngResource', 'ngDropzone',
        '720kb.tooltips',
        'crmApp.authentication', 'crmApp.account', 'crmApp.
            product', 'crmApp.contact', 'crmApp.lead', 'crmApp.
            opportunity'
    ])
```

V datoteki *config.js* so definirane poti oziroma stanja, podobno kot v *urls.py* pri DRF. Pri vsakem stanju je treba definirati URL, HTML predlogo (delni pogled) in nadzornika. V izseku programa (Koda 4.7) lahko vidimo še atributa *authenticate* in *resolve*. S prvim zahtevamo, da je uporabnik prija-

vljen, znotraj drugega pa povemo, katere knjižnice in gradnike naj aplikacija dodatno naloži ob prihodu na omenjeno stanje.

Koda 4.7: Primer definicije poti - datoteka *config.js*

```
.state('app.product.add', {
    url: "/add",
    templateUrl: "app/scripts/product/views/product-add.html",
    controller: 'ProductAddController',
    authenticate: true,
    resolve: {
        deps: ['$ocLazyLoad', function ($ocLazyLoad) {
            return $ocLazyLoad.load(['select', 'wizard', '
               inputMask', 'autonumeric',
                        'summernote', 'typehead', 'datepicker'
                           ], {
                    insertBefore: '#lazyload_placeholder'
                })
                .then(function () {
                    return $ocLazyLoad.load([
                        'app/scripts/product/services.js',
                        'app/scripts/product/controllers.js',
                        'app/scripts/account/services.js'
                    ]);
                });
        }]
    }
})
```

Omenjene dodatne JavaScript in CSS knjižnice smo definirali v datoteki *config.lazyload.js*. To uporablja knjižnica *ocLazyLoad*, ki služi lenemu nalaganju. Leno nalaganje (ang. lazy loading) je programerski pristop, kjer se komponente in odvisnosti naložijo šele, ko je to potrebno. Takšen pristop nalaganja in incializacije objektov omogoča hitrejše delovanje aplikacije. Uporaba pristopa je še posebej smiselna, kadar ima aplikacija veliko komponent, kar v našem primeru drži.

4.4.2 Lokalizacija

Ker želimo podpreti uporabo aplikacije za uporabnike iz različnih držav, moramo poskrbeti za prevode uporabniškega vmesnika. Uporabili smo preprosto knjižnico za lokalizacijo aplikacij AngularJS, imenovano *angular-gettext*. Naslove, oznake in ostala besedila, za katere želimo omogočiti prevode, smo v HTML kodi označili z direktivo *translate*. Nato smo nastavili Grunt opravilo, ki na izvedbo ukaza *grunt build* "izloči" vsa označena besedila in ustvari datoteko s končnico *.po*, pripravljeno za prevod (podobno kot pri strežniškem delu aplikacije).

Jezik nastavimo v datoteki *app.js*, glede na nastavitve uporabnika. Najprej nastavimo polje *Accept-Language* v glavi HTTP zahtevka, da bo storitev vračala rezultate v pravem jeziku, nato pa nastavimo še prevode uporabniškega vmesnika (Koda 4.8).

Koda 4.8: Nastavitev jezika na strani odjemalca

```
$http.defaults.headers.common['Accept-Language'] = $rootScope.
    globals.account.language;
// ...
gettextCatalog.setCurrentLanguage($rootScope.globals.account.
    language);
```

4.4.3 Komponente spletnega vmesnika

Aplikacijo bomo predstavili po sklopih, kot smo jo tudi razvijali. Predstavili bomo osnovne funkcionalnosti in poglede aplikacije ter poizkusili na hitro razložiti, kako kaj deluje. Pregledali bomo, kako se odjemalski del aplikacije povezuje na strežniški del in kako obdela podatke, ki jih od njega dobi oziroma mu jih posreduje.

4.4.3.1 Skupno (common)

Prvi sklop aplikacije smo poimenovali common. V njem so komponente, ki so skupne vsem modulom: pogledi, nadzorniki, direktive in filtri. Glavni del, ki je skupen vsem stranem, je izgled uporabniškega vmesnika.
Uporabniški vmesnik je razdeljen na več delov. Sestavljen je iz stranskega navigacijskega menija, vrhnje vrstice z uporabniškimi orodji, polja za prikaz vsebine, noge, dela za drobtine, in hitrega iskalnika. Vse te HTML predloge se nahajajo znotraj mape *views* v modulu common, skupaj pa so v celoto povezane v predlogi *app.html* (Koda 4.9).

Koda 4.9: Sestava posameznih delov strani v celoto - datoteka *app.html*

```
<!-- Sidebar -->
<div ng-include src="␣'app/scripts/common/views/blocks/sidebar.
    html'␣" include-replace>
</div>

<div class="page-container">
    <div ng-include="␣'app/scripts/common/views/blocks/header.
        html'␣"></div>

    <div class="page-content-wrapper">
        <div class="content">
            <div class="full-height␣full-width" ui-view></div>
        </div>
        <div ng-include="␣'app/scripts/common/views/blocks/
            footer.html'"></div>
    </div>

</div>

<!--Quickview -->
<div ng-include src="␣'app/scripts/common/views/blocks/
    quick_view.html'␣" include-replace></div>

<!-- Overlay -->
```

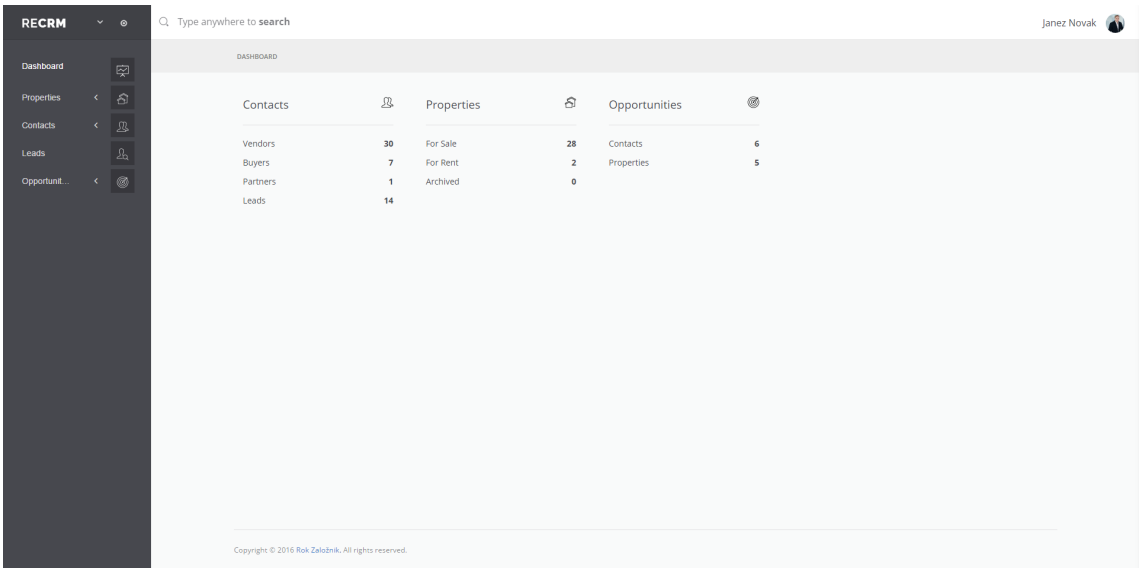
```
<div ng-include src="'app/scripts/common/views/blocks/
    quick_search.html'" include-replace></div>
```

Osnovna stran je definirana v datoteki *index.html* in se nahaja v korenski mapi aplikacije. Na začetku dokumenta z direktivo *data-ng-app* (Koda 4.10) povemo, da je ta dokument nadrejen vsem ostalim pogledom, z direktivo *ng-controller* pa, kateri nadzornik je s pogledom povezan. V preostalem delu dokumenta so vključene vse potrebne knjižnice JavaScript in CSS.

Koda 4.10: Direktivi *data-ng-app* in *ng-controller* - datoteka *index.html*

```
<!DOCTYPE html>
<html lang="en" data-ng-app="crmApp" ng-controller="
    AppController">
<head>
    <meta http-equiv="content-type" content="text/html;charset=
        UTF-8" />
    <meta charset="utf-8" />
```

V sklop skupno spada tudi domača stran aplikacije, ki jo imenujemo nadzorna plošča (ang. dashboard). Na tej strani so izpisani podatki o številu kontaktov, nepremičnin in priložnosti. Tako lahko agent hitro opazi morebitne nove priložnosti ali druge novosti (Slika 4.4).

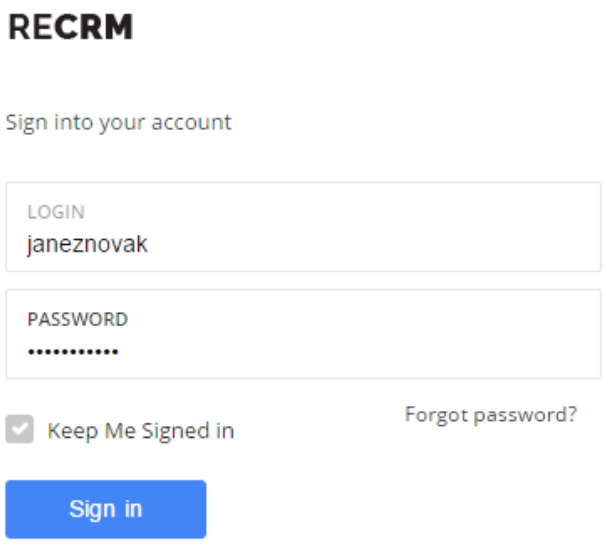


Slika 4.4: Prikaz stranskega menija in nadzorne plošče

4.4.3.2 Avtentikacija (authentication)

Naslednji pomemben vidik naše aplikacije je prijava uporabnika, saj se mora uporabnik za uporabo sistema obvezno prijaviti. Temu smo namenili celoten modul, ker moramo poleg avtentikacije uporabnika, poskrbeti tudi za avtorizacijo dostopov do različnih delov aplikacije. Registracije na tem mestu nismo razvili, saj mora novo organizacijo in vsaj enega uporabnika upravljalca (agenta) dodati skrbnik sistema.

Uporabnik mora za prijavo vnesti uporabniško ime in geslo (Slika 4.5). Ob kliku na gumb *Sign in* v nadzorniku aplikacije kličemo storitev, ki smo jo implementirali za avtentikacijo uporabnika. Ta končni točki API preko varne povezave (HTTPS) pošlje vnešena podatka. Spletna storitev preveri, če se uporabniško ime in geslo ujemata. V primeru da se, strežnik generira in shrani uporabniški žeton (ang. token) v podatkovno bazo ter nam ga v odgovoru vrne, skupaj z osnovnimi podatki uporabnika. Te podatke shranimo v lokalno shrambo brskalnika, da se ob morebitni osvežitvi strani ne



Slika 4.5: Prijava uporabnika v aplikacijo

izgubijo. Ker je tudi za vsak nadaljnji klic naše storitve potrebna avtentikacija, nastavimo še privzeto glavo "Authorization", z uporabniškim žetonom (Koda 4.11) v AngularJS objektu *$http*. S tem dosežemo, da se ob vsakem poslanem zahtevku na strežnik v glavo zahtevka HTTP avtomatsko doda polje, potrebno za avtentikacijo na strežniku.
Preostane nam le še, da uporabnika preusmerimo na domačo stran, kjer se v zgornji orodni vrstici aplikacije pojavita ime in prikazna slika uporabnika.

Koda 4.11: Dodajanje žetona v glavo HTTP zahtevka

```
$http.defaults.headers.common['Authorization'] = 'Token ' +
    $rootScope.globals.token;
```

Če se uporabniško ime in geslo ne ujemata, uporabniku prikažemo obvestilo o napaki.

4.4.3.3 Uporabnik (account)

Za prijavo uporabnika smo poskrbeli že v modulu authentication. V sklopu modula account, lahko uporabnik vidi in ureja svoj uporabniški profil. Spremeni lahko ime, priimek, licenčno številko, telefon, sliko profila, uporabniško ime, elektronsko pošto in geslo. Na stran za urejanje profila lahko pride tako, da klikne na gumb *Profile*. Do tega gumba pa pride tako, da v zgornji orodni vrstici klikne na sliko profila, ki se nahaja na desni strani zaslona, poleg svojega imena in priimka. Odpre se mu seznam, na katerem se poleg gumba za urejanja profila nahaja še gumb *Logout*, ki omogoča odjavo iz sistema.
Če ima agent pravice upravljalca, je na seznamu tudi gumb *Organization*. Ta pripelje uporabnika na stran, kjer lahko vidi in ureja tudi osnovne podatke o organizaciji ter njenih agentih. Obstoječe profile lahko ureja in jim spreminja dovoljenja. Lahko pa tudi kreira nove uporabniške profile ali obstoječe deaktivira.

Posamezne strani smo razvili posebej. Vsak pogled smo definirali v mapi *views* v svoji datoteki HTML (npr. *account-edit.html*). Vsak pogled je povezan s svojim nadzornikom, ti pa se nahajajo v datoteki *controllers.js*. Za urejanje profila smo definirali nadzornika *AccountEditController*.
Ker gre za urejanje, na začetku naložimo obstoječe podatke. Te navadno pridobimo s klicem ustreznega API-ja, vendar v tem primeru to ni potrebno. Podatke o uporabniku že imamo shranjene v lokalni shrambi brskalnika, saj smo jih pridobili med prijavo. Na tem mestu nastavimo spremenljivko *$scope.account*, s podatki iz lokalne shrambe.
Sedaj, ko imamo podatke o uporabniku, jih moramo ustrezno "naložiti" v obrazec za urejanje. Vsebino vsakega vnosnega polja določimo z direktivo *ng-model*. S tem povemo, s katero spremenljivko v kontekstu (*$scope*) je vrednost vnosnega polja povezana (Koda 4.12). Povezava deluje v obe smeri (ang. 2-way data binding).

Koda 4.12: Del obrazca in vnosno polje za urejanje imena uporabnika

```
<form name="forms.account" class="form-horizontal" role="form"
    novalidate="novalidate">
    <div class="form-group">
        <label class="col-sm-3 control-label required" for="
            firstName" translate>First name</label>
        <div class="col-sm-3">
            <input type="text" class="form-control" id="
                firstName" name="firstName"
                    ng-model="account.firstName" ng-required="
                        true">
            <label class="error"
                    ng-show="forms.account.firstName.$invalid &&
                        showErrors"
                    translate>This field is required.</label>
        </div>
```

Formata podatkov uporabnika, ki jih storitev vrača in ki jih sprejema za posodabljanje, se ne razlikujeta, zato dodatna obdelava ni potrebna. V nekaterih drugih nadzornikih, kjer je podatkov več, je pogosto potrebno formatirati podatke v ustrezno obliko JSON, da ustrezajo strežniški storitvi, ki se uporablja.

Uporabniški vmesnik in podatke imamo, potrebujemo le še akcijo, ki bo sprožila prenos podatkov na strežnik ter posodobitev uporabniškega profila. Zato smo na koncu vnosnega obrazca dodali gumb *Save*, s katerim uporabnik shrani podatke. Gumbu smo dodali direktivo *ng-click*, ki definira, katera funkcija v nadzorniku se kliče ob pritisku nanj. Funkcija preveri ustreznost obrazca, če so zahtevana polja izpolnjena, ter če so pravilna. V primeru, da ni napak, se kliče storitev *AccountService*, drugače pa se uporabniku prikaže obvestilo o napaki.

Storitev je v okolju AngularJS funkcija ali objekt za opravljanje specifičnih opravil in ga lahko uporabljamo po vsej aplikaciji, kar pripomore k organizaciji kode. Mi smo v večini primerov implementirali lastne storitve za komunikacijo s strežnikom, torej za pridobivanje ali shranjevanje podatkov preko naših storitev. *AccountService* smo podobno kot nadzornika definirali

v skupni datoteki *services.js*, znotraj modula *accounts* (Koda 4.13).

Koda 4.13: Storitev za uporabnika

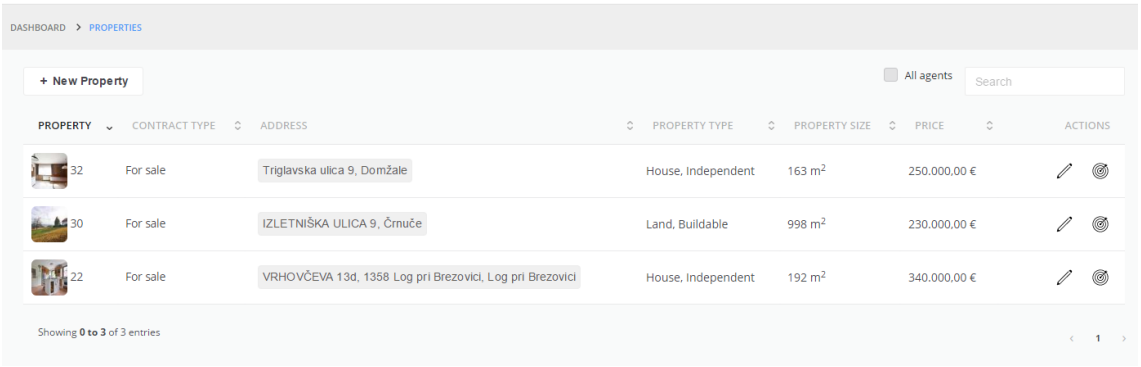
```
angular.module('crmApp.account')

    .factory('AccountService', ['$resource', '$rootScope',
        function ($resource, $rootScope) {
         return $resource($rootScope.API + '/accounts/:id', {id:
             '@id'}, {
             get: {
                 method: 'GET'
             },
             update: {
                 method: 'PUT'
             },
             delete: {
                 method: 'DELETE'
             },
             create: {
                 method: 'POST'
             }
         });
    }])
```

Storitev smo definirali z metodo *factory*, v kateri smo naredili objekt tipa *$resource*. To je vgrajena storitev, ki omogoča komunikacijo z RESTful storitvami. Podali smo mu pot do končne točke API za urejanje uporabnika in definirali metode za različne akcije (GET, PUT, DELETE, POST).

4.4.3.4 Produkt (product)

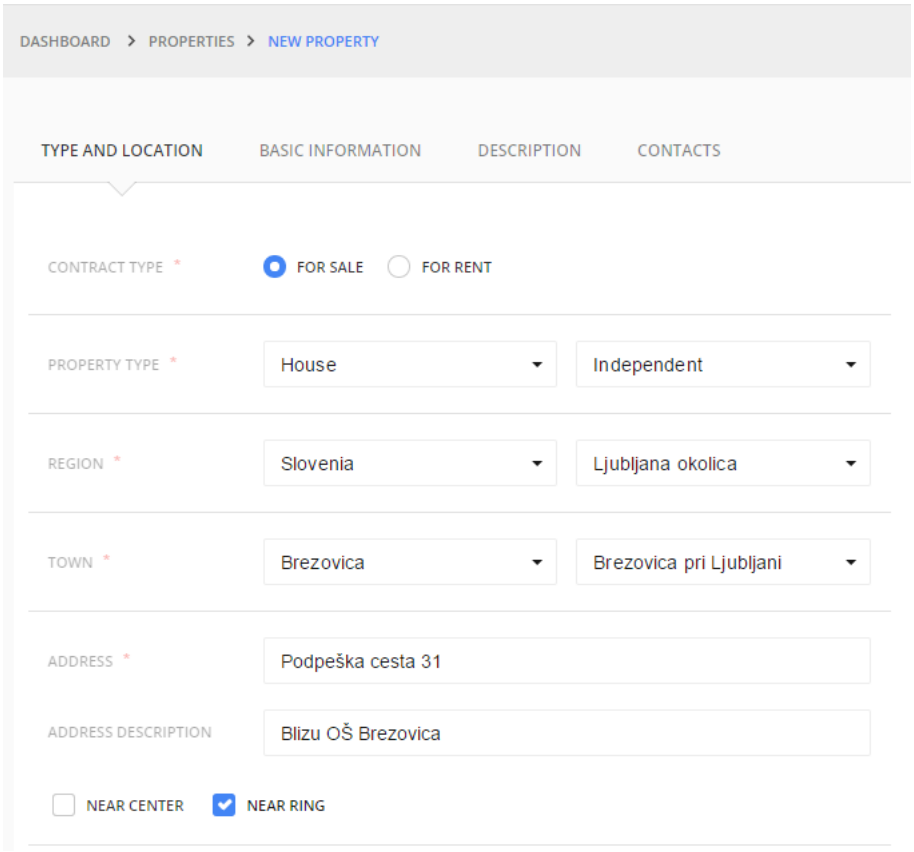
Uporabniki imajo možnost pregleda, dodajanja, brisanja in urejanja nepremičnin. Na stranskem navigacijskem meniju se ob kliku na zavihek *Nepremičnine* prikaže seznam nepremičnin. Iz seznama lahko uporabnik razbere najpomembnejše podatke o nepremičnini, za boljšo predstavo pa je zraven tudi slika. Seznam lahko tudi sortira ali filtrira po agentih ali pa uporabi iskalnik za hitrejše iskanje nepremičnine (Slika 4.6).



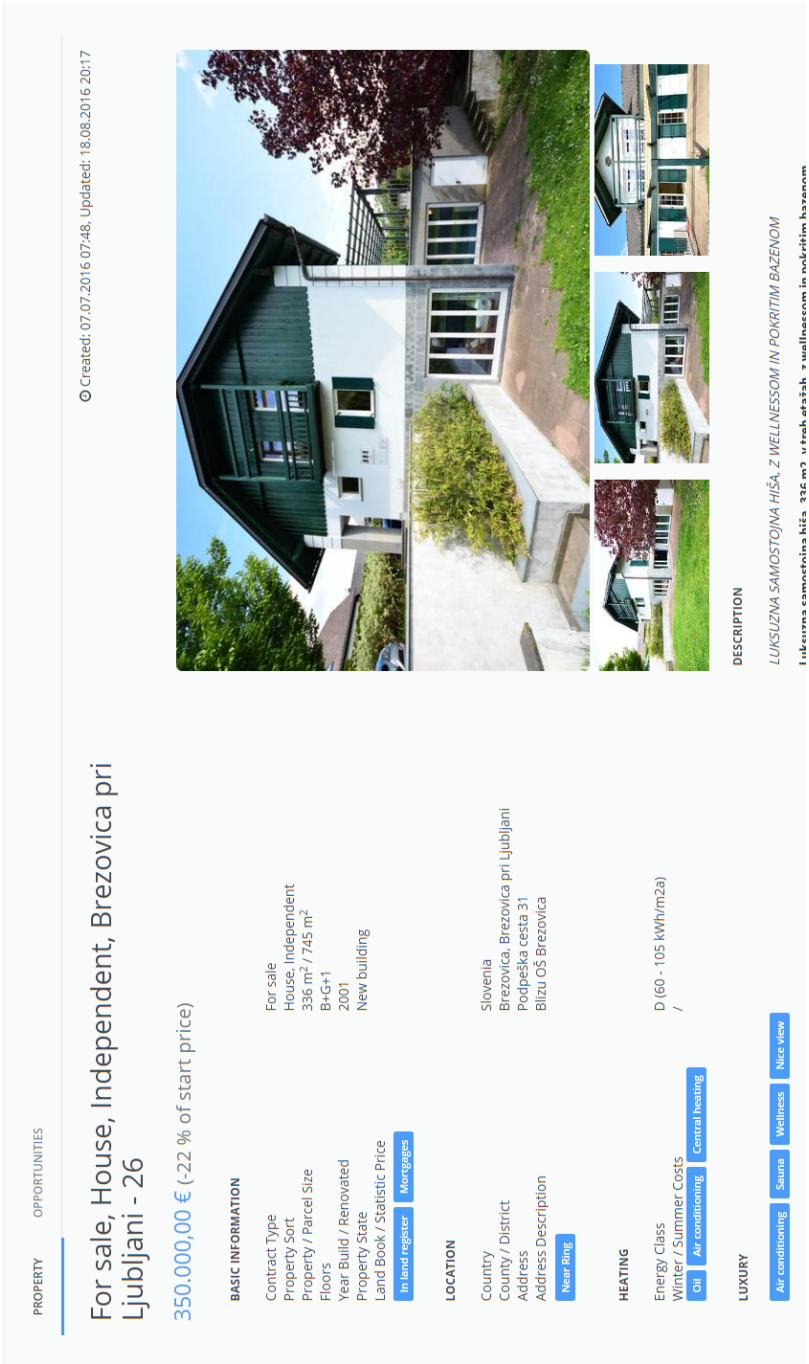
Slika 4.6: Seznam nepremičnin

Na skrajnem desnem koncu vrstice seznama sta dve uporabni bližnjici. Prva pripelje uporabnika neposredno na urejanje nepremičnine, druga pa na priložnosti povezane z nepremičnino. Za podroben pregled podatkov zadostuje klik kjerkoli na izbrani vrstici. V podrobnem pogledu lahko uporabnik hitro ureja ceno, če pa želi urediti še ostale podatke, pa to lahko stori s klikom na gumb *Edit*. S klikom na gumb *Deactivate* lahko nepremičnino deaktivira.

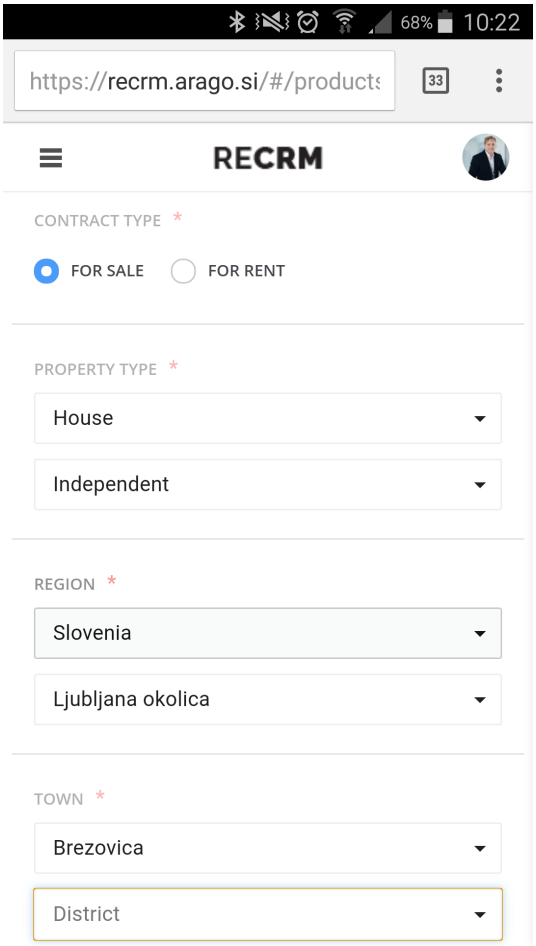
Dodajanje nove nepremičnine poteka tako, da uporabnik najprej klikne na gumb *New property*, ki je levo nad seznamom. Prikaže se obširen obrazec za vnos vseh podatkov, ki je razdeljen na več smiselnih sklopov, kar prispeva k večji preglednosti in doslednosti vnosa.
Najprej je potrebno vnesti podatke o vrsti in lokaciji nepremičnine (Slika 4.7). Temu sledijo osnovne informacije nepremičnine, kot so: cena, velikost, število nadstropij, leto izdelave in adaptacije, energijski razred, podatki o zemljiški knjigi, podroben opis, vrste ogrevanja, priključki, itd. V zadnjem sklopu so podatki o kontaktih (prodajalci) in podatki o pogodbi. Ko uporabnik obrazec potrdi, se mu prikaže še stran, na kateri lahko doda slike, nato pa s klikom na gumb *Save* dokonča postopek, aplikacija pa ga preusmeri na podroben pregled dodane nepremičnine (Slika 4.8). Nepremičnino lahko agent vnese tudi na terenu, preko mobilne naprave (Slika 4.9).



Slika 4.7: Dodajanje nepremičnine - vrsta in lokacija



Slika 4.8: Podroben pregled nepremičnine



Slika 4.9: Dodajanje nepremičnine na mobilnem telefonu

Koncept implementacije posameznih pogledov v sklopu produktov je podoben, kot smo ga opisali v prejšnjem podpoglavju pri uporabnikih. V datotekah HTML so definirani uporabniški vmesniki, v nadzornikih je poslovna logika in pretvorba podatkov za prikaz oziroma za shranjevanje, medtem ko je v storitvah koda za komunikacijo s strežnikom. Zaradi tega v nadaljevanju posebne pozornosti implementaciji posameznih pogledov ne bomo več namenjali in bomo raje opisali ter pokazali primere uporabe.
V tem sklopu imamo štiri osnovne poglede: seznam, podroben pregled, dodajanje in urejanje. Vsak pogled ima pripadajočega nadzornika.

ProductListController : je nadzornik, ki skrbi za seznam nepremičnin. S storitvijo pridobimo seznam podatkov. Ker je seznam lahko dolg, smo implementirali ostranjevanje. Za to smo spisali ustrezne funkcije, implementirali pa smo tudi funkcije za sortiranje in iskanje po seznamu.

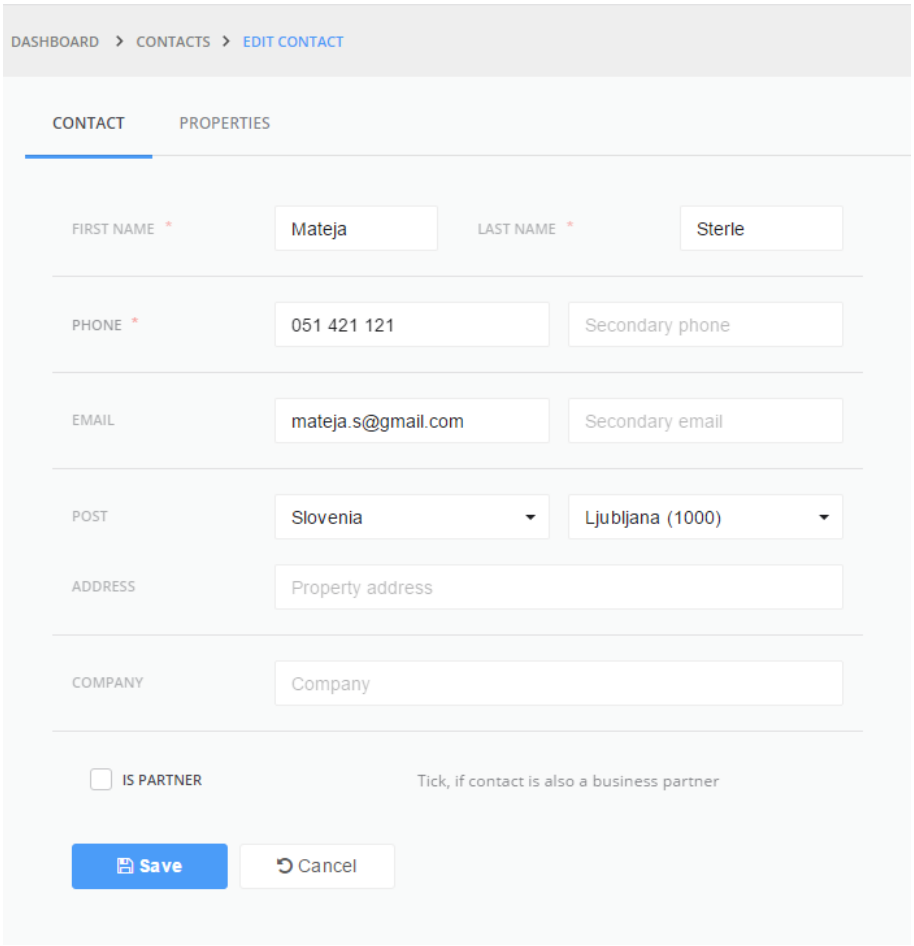
ProductDetailController : skrbi za podroben pogled nepremičnine. Njegova naloga je, da s pomočjo storitve dobi vse podatke o nepremičnini in jih spremeni v obliko, primerno za prikaz. Primarni ključ produkta, za katerega je potrebno prenesti podatke, dobimo kot parameter iz URL-ja.

ProductAddController : je odgovoren za dodajanje nove nepremičnine. Glavna funkcija je preverjanje pravilnosti vnosnega obrazca in pretvorba podatkov v obliko, primerno za klic storitve.

ProductEditController : je nadzornik za urejanje obstoječe nepremičnine. Podobno kot pri podrobnem pregledu iz URL-ja identificiramo produkt, ki ga želi uporabnik urediti. Preko storitve potem dobimo podatke in jih pretvorimo v ustrezno obliko, da napolnimo vnosna polja obrazca. Ko obrazec shranimo pa moramo podatke ponovno pretvoriti v primerno obliko.

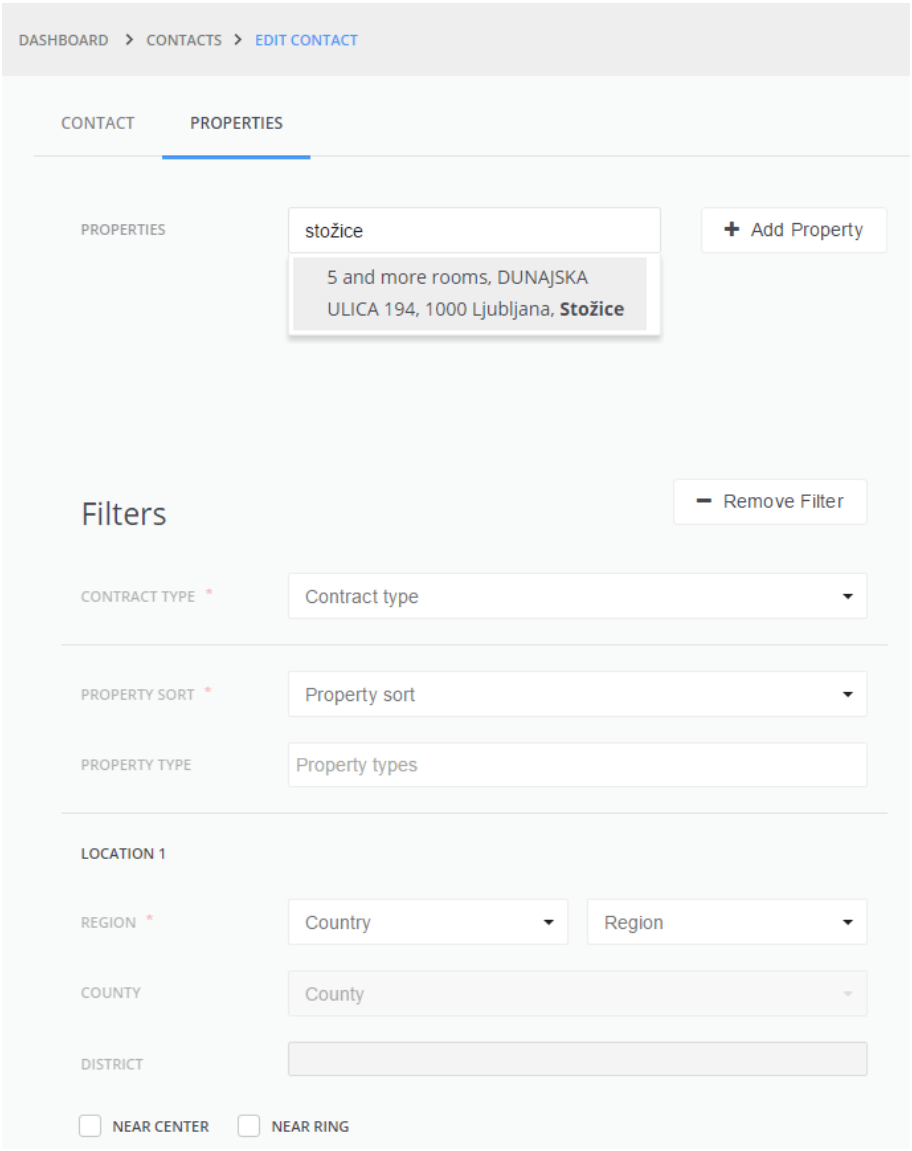
4.4.3.5 Kontakt (contact)

Kontakti so osebe, lahko bi jih imenovali tudi stranke, ki prodajajo ali kupujejo nepremičnino, oziroma oboje hkrati, lahko pa tudi nič od tega. Sklop je sestavljen, podobno kot pri produktih, iz seznama, podrobnega pogleda, urejanja in dodajanja. Pri dodajanju kontakta je v prvi fazi potrebno vnesti osnovne kontaktne podatke, obvezni so le ime, priimek in telefon (Slika 4.10). Ko uporabnik potrdi obrazec, se kontakt doda, aplikacija pa uporabnika preusmeri na urejanje kontakta. Tu lahko kontakt poveže z eno ali več nepremičninami, ki jih prodaja. V iskalno polje preprosto vnese poljubno lastnost nepremičnine, aplikacija pa vrne ustrezne rezultate, ki jih dobi s klicem storitve (Slika 4.11).



Slika 4.10: Urejanje osnovnih podatkov kontakta

Poleg tega pa lahko dodamo tudi iskalni filter, če nepremičnino iščemo. V iskalnem filtru lahko določimo vrsto pogodbe (prodaja ali/in najem), vrsto in tip nepremičnine (npr. hiša, samostojna ali hiša, vse vrste), nato določimo željene lokacije. Lahko izberemo celotno regijo v državi ali pa natančnejše področje (npr. Slovenija, Ljubljana mesto, center, Stara Ljubljana). Določimo lahko še cenovni razpon (cena od, cena do), stanje nepremičnine (opremljeno, delno opremljeno, neopremljeno) ter še nekaj drugih kriterijev.



Slika 4.11: Urejanje nepremičnin in dodajanje iskalnega filtra na kontaktu

Ko uporabnik shrani podatke, sistem takoj preveri, če za iskalni filter kontakta obstaja priložnost. Priložnosti lahko pogledamo v zavihku *Opportunities* v podrobnem pogledu kontakta ali v seznamu vseh priložnostih, do katerega lahko dostopamo preko stranskega navigacijskega menija.

4.4.3.6 Potencialna stranka (lead)

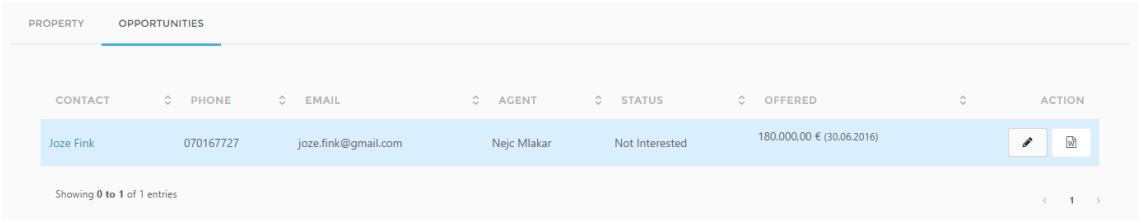
Potencialne stranke so osebe, ki prodajajo nepremičnino, a jih ne zastopajo agenti organizacije. Potencialne stranke lahko dodamo, urejamo, brišemo, lahko pa jih tudi spremenimo v kontakte. To storimo s klikom na gumb *Convert to contact*. Aplikacija s klicem ustrezne končne točke API z obstoječimi podatki osebe ustvari nov kontakt v podatkovni bazi. Do seznama potencialnih strank dostopamo preko stranskega navigacijskega menija. Seznam lahko sortiramo in po njem iščemo.

4.4.3.7 Priložnost (opportunity)

V stranskem navigacijskem meniju spodaj se nahaja gumb *Opportunities*. Ob kliku nanj se uporabniku prikaže seznam vseh priložnosti, pri katerem sta možna dva različna pogleda. Prvi je grupiran po produktih, drugi pa je grupiran po kontaktih, kar daje uporabniku izbiro pri pregledu priložnosti.

Privzeti je prvi pogled, torej skupine priložnosti urejene po produktih. Najprej je izpisana nepremičnina, pod njo pa je seznam kontaktov, katerih iskalni filtri se z njo ujemajo. Posamezna vrstica vsebuje osnovne podatke o kontaktu, kot so: ime, telefonska številka in elektronska pošta ter podatek o tem, kako je bil kontakt pridobljen. Na koncu vrstice, je izpisan še status priložnosti, cena, datum ter koliko in kdaj je kontakt nazadnje ponudil za nepremičnino. Te informacije lahko uporabnik ureja v zavihku *Opportunities* v podrobnem pregledu produkta. To lahko stori s klikom na gumb z ikono svinčnika, na desni strani seznama (Slika 4.12). Poleg urejanja, pa ima uporabnik tudi možnost prenosa pogodbe, čemur je namenjen gumb, ki se nahaja poleg gumba za urejanje. Ob kliku nanj aplikacija kliče končno točko API,

ki avtomatsko kreira Word dokument s podatki nepremičnine in izbranega kontakta. Uporabniku se odpre okno za prenos datoteke.



Slika 4.12: Seznam priložnosti na nepremičnini

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo na kratko pregledali kaj je strategija CRM in razložili probleme splošnih sistemov CRM. V nadaljevanju dela smo opisali izdelavo delujoče spletne aplikacije CRM, prilagojene za področje nepremičnin.
Cilj tega dela je izdelava aplikacije CRM za področje nepremčnin. Pomembno pa je tudi, da bo aplikacija uporabna in funkcionalna. Ker je delo nepremičninskih agentov, ki predstavljajo glavno ciljno skupino uporabnikov naše aplikacije predvsem tudi terensko, smo aplikacijo prilagodili tudi za uporabo na mobilnih napravah.
S trenutno verzijo aplikacije smo zadostili osnovnim funkcionalnostim za smiselno uporabo v praksi in dosegli zastavljene cilje. Implementirali smo sklope nepremičnin, kontaktov, potencialnih strank in priložnosti, ki nepremičninskim agentom omogočajo pregled nad nepremičninami, ki jih tržijo, in strankami, s katerimi poslujejo oziroma bodo poslovali. Aplikacija nepremičninskim agentom olajša tudi iskanje novih poslovnih priložnosti in delo z dokumenti, saj omogoča samodejno odkrivanje priložnosti ter kreiranje pogodbenih dokumentov z že izpolnjenimi podatki o nepremičnini in kontaktu.
Pred resno uporabo v praksi bo potrebno sistem še dodatno testirati. Do sedaj je bilo veliko testiranja že opravljenega, zaznane napake pa smo sproti odpravili. Kar nekaj preglavic smo imeli s pravilnim prikazom obrazcev in

drugih elementov strani na mobilnih napravah. Drugi problem, s katerim smo se srečali, pa je bilo počasno delovanje strežniškega dela aplikacije. Zaradi relativno kompleksnih entitet in relacij so nekatere bazne poizvedbe na gostečem strežniku trajale predolgo časa. Problem smo rešili z optimizacijo baznih poizvedb in dodatnim predpomnenjem na strani strežnika (ang. caching). Določeni hrošči in morebitne izboljšave se bodo ob dodatnem testiranju pravih uporabnikov zagotovo še razkrili, takrat jih bomo poizkusili odpraviti.

5.1 Nadaljnje delo

Trenutna različica aplikacije zadostuje osnovnim kriterijem za uporabo, vendar se zavedamo, da je možnosti za izboljšave takšnega tipa aplikacije še veliko. Potrebe po dodatnih funkcionalnostih bomo v prihodnosti najlažje zaznali s komunikacijo z dejanskimi uporabniki. Tisti, ki sistem uporabljajo in poznajo procese poslovanja, najbolje vedo, kaj potrebujejo in česa ne.
Nadaljni koraki bodo šli v smeri odprave napak in razvoja novih funkcionalnosti. Nekaj idej za nadaljnji razvoj že imamo:

- dodatno filtriranje seznama nepremičnin,
- implementacija beleženja aktivnosti,
- implementacija opozoril o novih priložnostih,
- pošiljanje kampanj elektronske pošte,
- implementacija koledarja, za lažje beleženje opravkov.

Literatura

[1] Mobile devices traffic searches exceeds computers. Dosegljivo: https://adwords.googleblog.com/2015/05/building-for-next-moment.html, 2015. [Dostopano: 1.7.2016].

[2] Customer relationship management. Dosegljivo: https://en.wikipedia.org/wiki/Customer_relationship_management, 2016. [Dostopano: 15.3.2016].

[3] Vineet Kumar and Werner Reinartz. *Customer relationship management: Concept, strategy, and tools*. Springer Science & Business Media, 2012.

[4] Upravljanje odnosov s strankami. Dosegljivo: https://sl.wikipedia.org/wiki/Upravljanje_odnosov_s_strankami, 2016. [Dostopano: 15.3.2016].

[5] Jan Manca. Analiza vrednotenja sistemov crm. Magistrsko delo, Ekonomska fakulteta, Univerza v Ljubljani, 2010.

[6] Django REST Framework. Dosegljivo: http://www.django-rest-framework.org/, 2016. [Dostopano: 10.4.2016].

[7] Django (web framework). Dosegljivo: https://en.wikipedia.org/wiki/Django_(web_framework), 2016. [Dostopano: 30.5.2016].

[8] PostgreSQL. Dosegljivo: https://en.wikipedia.org/wiki/PostgreSQL, 2016. [Dostopano: 30.5.2016].

[9] AngularJS. Dosegljivo: https://en.wikipedia.org/wiki/AngularJS, 2016. [Dostopano: 10.4.2016].

[10] Bootstrap (front-end framework). Dosegljivo: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework), 2016. [Dostopano: 2.6.2016].

[11] HTML5. Dosegljivo: https://en.wikipedia.org/wiki/HTML5, 2016. [Dostopano: 30.5.2016].

[12] Representational state transfer. Dosegljivo: https://en.wikipedia.org/wiki/Representational_state_transfer, 2016. [Dostopano: 30.7.2016].

[13] Single-page application. Dosegljivo: https://en.wikipedia.org/wiki/Single-page_application, 2016. [Dostopano: 6.6.2016].

[14] Django Documentation. Dosegljivo: https://docs.djangoproject.com/en/dev, 2016. [Dostopano: 10.4.2016].

[15] AngularJS - API Docs. Dosegljivo: https://docs.angularjs.org/api, 2016. [Dostopano: 10.4.2016].